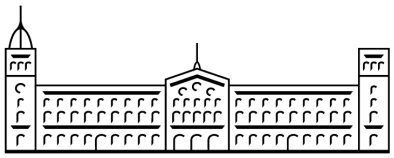


Trabajo final de grado

GRADO DE MATEMÁTICAS

Facultad de Matemáticas e Informática
Universitat de Barcelona

RESOLUCIÓN DE SISTEMAS LINEALES Y NO LINEALES

Autor: Noel Méndez Lison

Director:	**Montserrat Navarro**
Realizado en:	**Departamento de Matemática Aplicada y Análisis**
Barcelona,	**29 de junio de 2017**

Abstract

The problem of solving systems of equations is one of the oldest and with more applications in a variety of situations. A first example is finding a curve to which belong n fixed points (x_i, y_i), $1 \leq i \leq n$. For this, the usual procedure consists in searching for a polynomial of degree n, $p(z) = \sum_{i=0}^{n} a_i \cdot z_i$, such that $p(x_i) = y_i$. Imposing these conditions we find a linear system such that its solutions yield the values of a_i, so that this polynomial will be the desired curve. Another field in which the problem is used is in digital signal processing, an area in engineering dedicated to the analysis and processing of signals (audio, voice, image or video). The problem also appears in structural analysis, the resolution of equations of material endurance to find internal strain, deformation and internal tensions that happen in a given structure. Finally, where it is more frequently found is in the field of linear programming and non-linear problem approximation.

For all the above, we want to study different methods for solving equation systems. In the first section we show some preliminary questions related to the topic that will be needed in the sequel. In the second section we present some the more frequent methods, such as the direct methods (to which the gaussian methods belong), iterative methods and Krylov methods. Next we study the efficiency of these methods by analysing some of the most important aspects such as the error, running time and number of iterations needed for iterative and Krylov methods, all by the use of some examples. In the case of iterative methods we see how the spectral radius is a useful tool for these methods, specially for convergence issues. This is why in the third section some methods for finding eigenvalues are presented, the most important of which is the maximum modulus method. In the fourth section we present some of the most frequent methods for non-linear systems, such as Newton's methods for various variables or the continuation method. Finally, by the use of the results we have obtained, we reach various conclusions about the variety of methods studied in this piece of work.

Resumen

El problema de resolver sistemas de ecuaciones es uno de los problemas más antiguos y que más aplicaciones tiene en distintos ámbitos. Un primer ejemplo es encontrar una curva que pase por n puntos fijados (x_i, y_i), $1 \leq i \leq n$. Para ello se suele buscar un polinomio de grado n, $p(z) = \sum_{i=0}^{n} a_i \cdot z_i$, tal que $p(x_i) = y_i$. Imponiendo estas condiciones hallamos un sistema lineal cuyo resultado nos proporciona los valores de a_i por el cual dicho polinomio será la curva deseada. Otro campo donde se usa es en el procesamiento digital de señales, a la rama de la ingeniería que se dedica al análisis y procesamiento de señales (como el audio, voz, imagenes, video). También aparecen en el análisis estructural, en la resolución de las ecuaciones de la resistencia de materiales para encontrar los esfuerzos internos, deformaciones y tensiones que actúan sobre una estructura resistente. También el ánalisis dinámico estudiaría el comportamiento dinámico de dichas estructuras

y la aparición de posibles vibraciones para la estructura. Finalmente, donde más frecuentemente suele aparecer es en el campo de la programación lineal y en la aproximación de problemas no lineales.
Por todo ello, queremos estudiar distintos métodos de resolución de sitemas. En el primer tema, veremos algunos preliminares que necesitaremos a lo largo de este trabajo. En el segundo, estudiaremos los métodos más frecuentes para resolver un sistema lineal como los directos (del cual están los ya conocidos métodos gaussianos), iterativos y de Krylov. Luego, veremos la eficacia de estos métodos viendo alguno de los factores más importantes como el error, tiempo de ejecución y número de iteraciones para los métodos iterativos y de Krylov mediante algunos ejemplos. En el caso de los métodos iterativos, veremos como el radio espectral será una herramienta útil para dichos métodos, especialmente para la convergencia. Es por ello, que en el tercer tema, estudiaremos algunos métodos para encontrar los valores propios, especialmente el de módulo máximo. En el cuarto, veremos alguno de los métodos más frecuentes de los sistemas no lineales, como Newton en diversas variables o de continuación. Finalmente, y a la vista de lo que hemos ido obteniendo, concluiremos con una breve conclusión, los métodos vistos.

Agradecimientos

Quiero dar las gracias a mi tutora por el seguimiento de todos estos meses, a mis familiares por el apoyo que me han dado y a mis compañeros, en especial a Miquel por la ayuda del abstract.

Índice

Índice **4**

1. Preliminares **6**

1.1. Normas vectoriales y matriciales 6

1.2. Estructura de matrices 9

2. Métodos númericos de sistemas lineales **12**

2.1. Métodos directos 14

2.1.1. Método de Gauss 14

2.1.2. Gauss con pivotaje (Parcial maximal por columnas) 15

2.1.3. Gauss con pivotaje (Total) 15

2.1.4. Descomposición LU 16

2.1.5. Descomposición LU con pivotaje 16

2.1.6. Variantes de la descomposición LU 17

2.1.7. Descomposición Cholesky 18

2.1.8. Descomposición QR 19

2.2. Métodos iterativos 21

2.2.1. Jacobi 22

2.2.2. Gauss-Seidel 22

2.2.3. Sobrerelajación (SOR) 23

2.3. Métodos de Krylov 24

2.3.1. Minimización 24

2.3.2. Gradiente conjugado 25

2.3.3. Gradiente biconjugado 26

2.4. Error en la resolución y comparación de los métodos 26

2.5. Tiempo de ejecución y cálculo de operaciones 69

2.6. Error iterativo y error de discretización 73

2.7. Cota error y condición matriz 78

3. Métodos numéricos de calculo de valores propios **82**

3.1. Método de la potencia 82

3.2. Variantes del método de la potencia 83

3.2.1. Método de la potencia inversa 83

3.2.2. Método de la potencia desplazada 84
3.2.3. Aplicación 84
3.3. Iteración QR 85
3.4. Descomposición en valores singulares (SVD) 85
3.5. Arnoldi 86
3.6. GMRES (Generalized Minimum Residual) 87
3.7. Lanczos 87
3.8. Aplicaciones 89
3.8.1. Potencia 89
3.8.2. GMRES 91
3.8.3. Lanczos 92
3.9. Comentario 92

4. Métodos numéricos de sistemas no lineales 94
4.1. Métodos de Newton en diversas variables 94
4.1.1. Caso no lineal 94
4.1.2. Caso lineal 95
4.2. Métodos de continuación 97
4.2.1. Caso no lineal 97
4.2.2. Caso lineal 98

5. Conclusiones 99

Referencias 103

1. Preliminares

1.1. Normas vectoriales y matriciales

Sea V un espacio vectorial sobre $\Re^n$ y sea $E = \mathcal{M}_{n\times n}$, donde $\mathcal{M}_{n\times n}$ indica el conjunto de matrices reales con n filas y n columnas.

Definición 1.
Una norma vectorial sobre V es una aplicación $||\ ||: V \longrightarrow \Re$ tal que cumpla:

1. $||x|| \geq 0\ \forall x \in V$
2. $||x|| = 0 \Leftrightarrow x = 0$
3. $||c \cdot x|| = c \cdot ||x||\ \forall c \in \Re\ y\ \forall x \in V$
4. $||x + y|| \leq ||x|| + ||y||\ \forall x, y \in V$

Definición 2.
Una norma matricial sobre E es una aplicación $||\ ||: E \longrightarrow \Re$ tal que cumpla:

1. $||A|| \geq 0\ \forall A \in M$
2. $||A|| = 0 \Leftrightarrow A = 0$
3. $||c \cdot A|| = c \cdot ||A||\ \forall c \in \Re\ y\ \forall A \in M$
4. $||A + B|| \leq ||A|| + ||B||\ \forall A, B \in M$
5. $||A \cdot B|| \leq ||A|| \cdot ||B||\ \forall A, B \in M$

Definición 3.
Dada una norma vectorial sobre V, $||\ ||_V$, se puede construir una norma matricial sobre M, $||\ ||_M$, definida por:

$$||A||_M = \max_{||x||_V=1} ||A \cdot x||_V$$

llamada norma matricial subordinada que es consistente (es decir $||A \cdot x||_V \leq ||A||_M \cdot ||x||_V\ \forall x \in V\ y\ \forall A \in M$).

Hay sistemas de ecuaciones que son muy sensibles a los errores en los datos o en los cálculos del método; por lo que al resolverlos producen resultados con mucha variabilidad. Por ejemplo el sistema:

$$\begin{pmatrix} 2 & 4 & 5 \\ 6 & 9 & 8 \\ 4{,}1 & 5 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 220 \\ 490 \\ 274 \end{pmatrix}$$

tiene como solución $(40, 10, 20)$ mientras que el sistema:

$$\begin{pmatrix} 2 & 4 & 5 \\ 6 & 9 & 8 \\ 4.2 & 5 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 220 \\ 490 \\ 274 \end{pmatrix}$$

tiene como solución $(20, 31.53, 10.76)$. Así, un cambio menor en un coeficiente produjo un cambio muy significativo en la solución, es decir que el resultado fue afectado fuertemente por este cambio producido por un posible error de medida (de 4.1 a 4.2 como elemento a_{31} y el resto igual). Es por ello que se dice que dicha matriz está mal condicionada. Para saber si una matriz está mal condicionada o no, necesitamos una definición previa:

Definición 4.
Sea $||\ ||$ una norma matricial y sea A una matriz no singular. Llamaremos número de condición de la matriz A al número

$$\kappa(A) = ||A|| \cdot ||A^{-1}||$$

Observación 1.
Hallar la inversa de una matriz además de costoso puede ser muy inestable; por este motivo, calcular el $\kappa(A)$ por la propia definición no sería muy recomendable. Es por ello que existen algunos métodos para calcular dicho número sin hallar la inversa. Uno de estos métodos es el método de Hager. Dicho método es para encontrar la condición de la matriz en norma 1. El método consta de los siguientes pasos:

1. *Se inicializa $\rho = 0$ y $b = \begin{pmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}^t$.*
2. *Resolver $A \cdot x = b$.*
3. *Si $||x||_1 \leq \rho$, ir al 7. En caso contrario, $\rho = ||x||_1$ e ir al 4.*
4. *Resolver $A^t \cdot z = y$, donde $y_i = \begin{cases} 1 & : x_i \geq 0 \\ -1 & : x_i < 0 \end{cases}$.*
5. *Sea z_j la componente con el valor más grande en valor absoluto.*
6. *Si $|z_j| > z^t \cdot b$, escribiremos por b el vector donde sus componentes valen cero excepto la componente j-ésima donde vale uno y volveremos al 2. En caso contrario, ir al 7.*
7. *$\bar{\kappa_1}(A) = \rho \cdot ||A||_1$*

Definición 5.
Se dice que una matriz A esta bien condicionada si $\kappa(A)$ es cercano a 1 y mal condicionada si $\kappa(A)$ se aleja significativamente de 1.

Observación 2.
Saber el $\kappa(A)$ será de utilidad a la hora de resolver un sistema pues un sistema con una matriz mal condicionada nos inducirá problemas númericos, de manera que solución hallada pueda no tener relación con la real. Sin embargo, que una matriz esté bien condicionada no implica que la solución dada sea precisa.

Propiedad 1.
$1 \leq \kappa(A)$

Demostración 1.
Sea $||\ ||$ una norma matricial cualquiera. Entonces:
$1 = ||Id|| = ||A \cdot A^{-1}|| \leq ||A|| \cdot ||A^{-1}|| = \kappa(A)$ *(donde en la desigualdad hemos aplicado la propiedad 5. de la definición de norma matricial)*

Definición 6.
Se define el radio espectral de la matriz B por:

$$\rho(B) = \text{máx}\,\{|\lambda_i| \text{ donde } \lambda_i = \text{valor propio de la matriz } B\}$$

Propiedad 2.
Sea $||\ ||_V$ una norma vectorial y sea $||\ ||_M$ la norma subordinada de $||\ ||_V$. Entonces:

$$\rho(A) \leq ||A||_M$$

Demostración 2.
Sea λ el valor propio de módulo máximo de vector propio v tal que $||v||_V = 1$. Así $\lambda \cdot v = A \cdot v$. Tomando normas:

$$|\lambda| = |\lambda| \cdot 1 = |\lambda| \cdot ||v||_V = ||\lambda \cdot v||_V = ||A \cdot v||_M \leq ||A||_M \cdot ||v||_V = ||A||_M \cdot 1 = ||A||_M$$

Propiedad 3.
Sea $A \in \mathcal{M}_{m\times n}$ (donde $\mathcal{M}_{m\times n}$ indica el conjunto de matrices reales con m filas y n columnas) una matriz. Se puede demostrar que:

1. $||A||_1 = \text{máx}_{1\leq j\leq n} \sum_{i=0}^{m} |a_{ij}|$
2. $||A||_\infty = \text{máx}_{1\leq i\leq m} \sum_{j=0}^{n} |a_{ij}|$
3. $||A||_2 = \rho(A^* \cdot A)^{\frac{1}{2}}$ *(donde A^* es la matriz resultante de transponer y conjugar la matriz A)*

Teorema 1.
Sea A una matriz hermitiana. Si $\kappa_2(A)$ indica el número de condición de la matriz A con la norma matricial $||A||_2$, λ_1 es el valor propio de módulo mínimo y λ_n es el valor propio de módulo máximo, entonces:

$$\kappa_2(A) = \left|\frac{\lambda_n}{\lambda_1}\right|$$

Demostración 1.
Sea λ un valor propio de $A \Rightarrow A \cdot x = \lambda \cdot x$ (1). Así:

$$A^2 \cdot x = A \cdot A \cdot x \overset{(1)}{=} A \cdot \lambda \cdot x = \lambda \cdot A \cdot x \overset{(1)}{=} \lambda \cdot \lambda \cdot x = \lambda^2 \cdot x \Rightarrow A^{-2} \cdot x = \frac{1}{\lambda^2} \cdot x$$

Es decir, si λ es un valor propio de A entonces λ^2 y $\frac{1}{\lambda^2}$ es un valor propio de A^2 y A^{-2} respectivamente.

Por tanto si λ_1, λ_2, λ_3, $\cdots$, λ_n *son los valores propio de* A *tal que* $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq \cdots \leq |\lambda_n|$ *entonces*

$$\lambda_1^2 \leq \lambda_2^2 \leq \lambda_3^2 \leq \cdots \leq \lambda_n^2$$

lo cual implica

$$\frac{1}{\lambda_n^2} \leq \frac{1}{\lambda_{n-1}^2} \leq \cdots \leq \frac{1}{\lambda_3^2} \leq \frac{1}{\lambda_2^2} \leq \frac{1}{\lambda_1^2}$$

En resumen, hemos visto que si A *tiene valor propio de módulo máximo* λ_n *y valor propio de módulo mínimo* λ_1 *entonces* A^2 *y* A^{-2} *tienen valores propios de módulo máximo* λ_n^2 *y* $\frac{1}{\lambda_1^2}$ *respectivamente* $\implies$ $\rho(A^2) = \lambda_n^2$ *y* $\rho(A^{-2}) = \frac{1}{\lambda_1^2}$ (2).
Por otro lado,
A *matriz hermitiana* $\implies$ A *y* A^{-1} *matrices hermitianas* $\implies$ $A = A^*$ *y* $A^{-1} = A^{-1^*}$ (3)
Para finalizar la demostración:

$$\kappa_2(A) = ||A||_2 \cdot ||A^{-1}||_2 = \rho(A^* \cdot A)^{\frac{1}{2}} \cdot \rho(A^{-1^*} \cdot A^{-1})^{\frac{1}{2}} \overset{(3)}{=} \rho(A \cdot A)^{\frac{1}{2}} \cdot \rho(A^{-1} \cdot A^{-1})^{\frac{1}{2}} =$$

$$= \rho(A^2)^{\frac{1}{2}} \cdot \rho(A^{-2})^{\frac{1}{2}} \overset{(2)}{=} (\lambda_n^2)^{\frac{1}{2}} \cdot (\frac{1}{\lambda_1^2})^{\frac{1}{2}} = |\lambda_n| \cdot \left|\frac{1}{\lambda_1}\right| = \left|\frac{\lambda_n}{\lambda_1}\right|$$

Observación 3.
Así pues, diremos que una matriz A*, está bien condicionada si* λ_1 *es muy próximo a* λ_n *y que está mal condicionada si* λ_1 *es lejano a* λ_n*.*

Teorema 2.
Las normas $||\ ||$ *son equivalentes, es decir que si una norma cumple una de las propiedades que queremos, cualquier otra norma también lo cumplirá.*

1.2. Estructura de matrices

En esta sección consideraremos A una matriz compleja con n filas y n columnas. Entre paréntesis pondremos el caso en que A sea una matriz real con n filas y n columnas.

Definición 7.
Una matriz A *es regular si su determinante es distinto de* 0 *o lo que es lo mismo si existe su inversa, es decir si existe una matriz* B *tal que* $A \cdot B$ *es la identidad (normalmente la matriz* B *se denota por* A^{-1} *y en tal caso es única).*

Definición 8.
Una matriz A *es singular si no es regular.*

Definición 9.
Una matriz A *es hermitiana (simétrica) si* $A^* = A$ *(*$A^t = A$*).*

Definición 10.
Una matriz A *es unitaria (ortogonal) si* $A^* = A^{-1}$ *(*$A^t = A^{-1}$*).*

Definición 11.
Una matriz A es definida positiva si A es hermitiana y $x^* \cdot A \cdot x > 0$ (A es simétrica y $x^t \cdot A \cdot x > 0$) $\forall x \in \Re^n \setminus 0$.

Definición 12.
Una matriz A es banda (p, q) si $a_{ij} = 0$ con $i \geq j + p$ o $j \geq i + q$.

Definición 13.
Una matriz A es triangular superior si es una matriz banda $(1, n)$.

Definición 14.
Una matriz A es triangular inferior si es una matriz banda $(n, 1)$.

Definición 15.
Una matriz A es Hessenberg superior si es una matriz banda $(2, n)$.

Definición 16.
Una matriz A es Hessenberg inferior si es una matriz banda $(n, 2)$.

Definición 17.
Una matriz A es diagonal si es una matriz banda $(1, 1)$.

Definición 18.
Una matriz A es tridiagonal si es una matriz banda $(2, 2)$.

Definición 19.
Una matriz A es pentadiagonal si es una matriz banda $(3, 3)$.

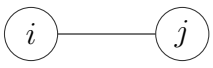
Definición 20.
Una matriz A es escasa o dispersa si muchos de sus elementos son cero.

Definición 21.
Dada una matriz A se define el semiancho de banda A como:

$$\beta = \max_{1 \leq i,j \leq n} (|i - j| \text{ con } a_{ij} \neq 0)$$

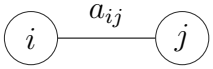
Reducción del semiancho de banda:
Se puede reducir el número del semiancho de banda de una matriz escasa pasando la matriz correspondiente a un grafo de la siguiente manera:

- Si la matriz A es simétrica entonces:

  $a_{ij} \neq 0 \iff$ 

- Si la matriz no es simétrica en vez de la línea de i a j, se une i y j mediante una flecha $\longrightarrow$ que vaya de i a j si $a_{i,j}$ es diferente de cero pero $a_{j,1}$ es 0 y se une i y j mediante una flecha $\longleftarrow$ que vaya de j a i si $a_{i,j}$ es diferente de cero pero $a_{j,1}$ es 0. En caso que tanto a_{ij} como a_{ji} sean cero uniremos i y j mediante una flecha $\longleftrightarrow$.

Así se puede considerar una matriz A, pasarla a un grafo y reordenar los nodos empezando por el nodo adecuado (aunque no siempre es fácil elegir el nodo adecuado) para que cuando se vuelva a pasar a matriz el semiancho de banda disminuya respecto a la matriz de entrada. Sería adecuado nombrar las aristas (a la línea o a la flecha según el caso) con el valor a_{ij} para tener presente su valor, es decir

$a_{ij} \neq 0 \iff$ 

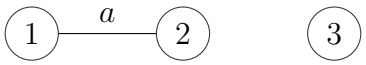
Este algoritmo se conoce como Cuthill-McKee.

Ejemplo 1.
Sea $a \in \Re$, consideremos la matriz:

$$A = \begin{pmatrix} 1 & 0 & a \\ 0 & 2 & 0 \\ a & 0 & 3 \end{pmatrix}$$

Su grafo asociado es:

Reordenando es:



Y finalmente, la matriz de este último grafo es:

$$B = \begin{pmatrix} 1 & a & 0 \\ a & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Tal y como observamos la matriz original A, el valor del semiancho de banda era de 2 mientras que la matriz final B el valor del semiancho de banda es de 1, reduciéndose así en 1.
***Nota:** En este ejemplo la reducción es mínima porque la dimensión de la matriz es pequeña pero hay casos en que la diferencia puede ser muy grande, por ejemplo si la matriz A hubiese sido $n \times n$ (con n grande) con todos los elementos ceros excepto a_{ii} que toma como valor i y a_{1n} y a_{n1} que toma como valor a, entonces el valor del semiancho de banda se reduce de $n-1$ a 1 (véase que dicho ejemplo corresponde con $n = 3$).*

2. Métodos númericos de sistemas lineales

Dado un sistema lineal $A \cdot x = b$, donde

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \in \Re^{n \times m}$$

es una matriz $n \times m$, (donde n indica el número de condiciones y m el número de incógnitas, en términos de matriz, indica el número de filas y el número de columnas respectivamente),

$$b = \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix}^t$$

es un vector columna n-dimensional denominado vector de términos independiente del sistema y finalmente,

$$x = \begin{pmatrix} x_1 & x_2 & \cdots & x_m \end{pmatrix}^t$$

es otro vector columna m-dimensional denominado vector de incógnitas del sistema, cuyos valores queremos calcular, de manera que se cumpla $A \cdot x = b$. Sea

$$\widehat{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2m} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} & b_n \end{pmatrix}$$

la matriz ampliada.
También escribiremos

$$A_k = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1k} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2k} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & a_{k3} & \cdots & a_{kk} \end{pmatrix}$$

el menor de orden k de la matriz A para $1 \leq k \leq n$
Hay dos tipos de sistemas:

1. **Sistemas incompatible:**
   Son los sistemas tales que $rang(A) \neq rang(\widehat{A})$. En este caso, el sistema no tendrá solución, es decir que no existirá el vector x tal que $A \cdot x$ sea b.

2. **Sistemas compatible:**
   Son los sistemas tales que $rang(A) = rang(\widehat{A})$. Estos sistemas tendrán, como mínimo, una solución, es decir existirá al menos un vector x tal que $A \cdot x$ sea b. Hay de dos tipos:

2.1. **Sistemas compatible determinado:**
Son los sistemas tales que $rang(A) = rang(\widehat{A}) = m$. Así, el sistema tendrá una única solución, es decir existirá un único vector x tal que $A \cdot x$ sea b. En caso que $n = m$, está condición equivale a que el determinante de A sea distinto de 0.

2.2. **Sistemas compatibles indeterminados:**
Son los sistemas tales que $rang(A) = rang(\widehat{A}) = r < m$. En este caso, el sistema tendrá más de una solución. Por lo general, en este tipo de ecuaciones, r incógnitas vendrán dada por las $m - r$ restates.

Observación 4.
Si b es el vector cero, es decir el vector que tiene por componentes $b_i = 0$, $1 \leq i \leq n$, entonces el sistema siempre es compatible puesto que siempre el vector cero será solución del sistema.

Si $n = m$ se dice que el sistema es cuadrado. En este caso, los sistemas podrían pertenecer a cualquiera de los tres tipos mencionados. Si el sistema es compatible, se dice que x es solución si $A \cdot x$ es b o lo que es lo mismo que la norma de $A \cdot x - b$ es cero.
Si $n < m$ se dice que el sistema esta infradeterminado. Este tipo de sistemas suelen ser compatibles indeterminados.
Finalmente, si $n > m$ se dice que el sistema esta sobredeterminado. En este caso, los sistemas suelen ser incompatibles. Para dicho sistema se definiría la solución x por

$$x = \min_{y \in \Re^m} ||A \cdot y - b||_2$$

Para encontrar dicho mínimo se suele usar el método de los mínimos cuadrados. Este método consiste en multiplicar por A^t

$$A \cdot x = b \implies A^t \cdot A \cdot x = A^t \cdot b$$

Y debemos resolver el sistema $M \cdot x = c$, donde $M = A^t \cdot A$ y $c = A^t \cdot b$. Disponemos de un sistema con el mismo número de ecuaciones que de incógnitas (m).

Teorema 3.
Es equivalente encontrar el mínimo de $\min_{y \in \Re^m} ||A \cdot y - b||_2$ *que encontrar la solución de las llamadas ecuaciones normales* $A^t \cdot A \cdot x = A^t \cdot b$.

Para encontrar la solución del sistema correspondiente, se dispone de varios tipos de métodos (directos, iterativos, de Krylov, de continuación,...). Muchos de estos métodos (aunque no todos) requieren que la matriz sea cuadrada, por tanto, empezaremos estudiando los sistemas con $m = n$. En los tres primeros apartados explicaremos algunos métodos para resolver estos sistemas. Mientras que en el primero nos concentraremos en los métodos directos; en el segundo y tercero lo haremos con los iterativos y de Krylov respectivamente. En el cuarto estudiaremos la eficacia de estos métodos a fondo mediante algunos ejemplos, dos de ellos serán concretos y

se conocerá la solución real mientras que los otros serán sistemas aleatorios y por tanto no sabremos la solución real. Finalmente, en el quinto, haremos un estudio de estos métodos más a fondo, es decir, miraremos el tiempo de ejecución y contaremos el número de operaciones dependiendo de la dimensión del sistema.

2.1. Métodos directos

Estos métodos consisten en transformar el sistema original $A \cdot x{=}b$ en otro equivalente, $M \cdot z{=}v$, que tenga la misma solución, pero que esta se pueda encontrar fácilmente, normalmente despejando las incógnitas. Así pues, son métodos con un número de pasos u operaciones finitas. Es por ello que, en un principio parece lógico pensar que la solución hallada, sería exacta salvo por errores numéricos en la matriz o errores de redondeo en el proceso.

2.1.1. Método de Gauss

El objetivo de este primer método es conseguir una matriz triangular superior para facilitar la resolución del sistema. Para ello, utilizamos combinaciones lineales de filas, que como sabemos no modifican la solución si dichas combinaciones también se aplican al vector de términos independientes. Con el multiplicador

$$m_{ij} = \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}} \quad \text{con} \quad \begin{cases} j = 1, \cdots, n-1 \\ i = j+1, \cdots, n \\ a_{ij}^{(1)} = a_{ij} \end{cases}$$

(donde $a_{ij}^{(j)}$ es el elemento de la nueva matriz que se obtiene cuando se ha hecho cero los elementos por debajo de la diagonal de la columna j-ésima) se puede conseguir hacer cero el elemento a_{ij} restando a los elementos de la fila i los elementos de la fila j multiplicado por m_{ij} de la matriz A, es decir para $0 < k < n$,

$$a_{ik}^{(j+1)} = a_{ik}^{(j)} - m_{ij} \cdot a_{jk}^{(j)}$$

Para que no afecte a la solución, este proceso se ha de hacer también para el vector b, es decir restar a la componente i-ésima, la componente j-ésima multiplicado por m_ij,

$$b_i = b_i - m_{ij} \cdot b_j$$

Este proceso se repite hasta hacer cero todos los elementos por debajo de la diagonal principal.
Una vez hayamos transformado la matriz A en una matriz equivalente y triangular superior U, la solución se calcula empleando el método de sustitución hacia atrás, es decir

$$\begin{cases} x_n = \frac{b_n}{u_{nn}} \\ \\ x_l = \frac{b_l - \sum_{l<k\leq n} u_{lk} \cdot x_k}{u_{ll}} \ ; \ \ l = n-1, ..., 1 \end{cases}$$

Un problema de este método es la posibilidad que en algún paso al dividir por $a_{jj}^{(j)}$ este sea cero o un número muy pequeño en valor absoluto (que sería como dividir prácticamente entre cero) lo que conlleva a problemas numéricos al buscar el multiplicador correspondiente. Uno de los casos más frecuentes que ocurre esto, es cuando la matriz está mal condicionada y esto conduce a errores en la solución. Para solventar dicho problema intentaremos aplicar Gauss con pivotaje para hallar una mejora.

2.1.2. Gauss con pivotaje (Parcial maximal por columnas)

El objetivo de este método es solucionar el problema que teníamos con el método anterior. Igual que antes el objetivo de este método es transformar la matriz inicial en una matriz triangular superior con la diferencia que ahora usaremos pivotaje parcial. Esto, quiere decir, que antes de calcular los multiplicadores correspondientes, se intercambiarán la fila principal i por la fila k-ésima, donde k es tal que

$$a_{ki} = \max_{i \leq j \leq n} |a_{ji}|$$

tanto de la matriz A como del vector de términos independientes para que no afecte a la solución. De esta forma, se evita dividir entre un número muy pequeño o cero.

2.1.3. Gauss con pivotaje (Total)

El objetivo de este método es mejorar el problema que teníamos anteriormente. Igual que antes el objetivo de este método es transformar la matriz inicial en una matriz triangular superior con la diferencia que ahora usaremos pivotaje total en vez de parcial. Esto, quiere decir, que antes de calcular los multiplicadores correspondientes, se intercambiarán la fila principal i por la fila k-ésima de A, la componente i-ésima por la componente k-ésima del vector de términos independientes b, la columna i por la columna r-ésima de A y la componente i-ésima por la componente r-ésima del vector de incógnitas x, donde k y r son tales que

$$a_{kr} = \max_{i \leq j,s \leq n} |a_{js}|$$

para que no afecte a la solución. De esta forma, se evita dividir entre un número muy pequeño o cero. Si en un paso i todos los a_{jl} (donde $i < j, l \leq n$) son más pequeños que una cierta tolerancia se terminaría el proceso. De esta manera, tenemos dos opciones:

1. Si $b_k = 0 \ \forall k \geq i$ el sistema tiene infinitas soluciones es decir estamos en el caso de un sistema compatible indeterminado.

2. En caso contrario, tendríamos una ecuación donde un lado de la igualdad es 0 y al otro lado un real distinto de 0. Como dicha ecuación no tiene solución, el sistema inicial tampoco lo tendrá por lo que tendríamos un sistema incompatible.

2.1.4. Descomposición LU

El objetivo de este método es descomponer la matriz A en el producto de dos matrices L y U, donde L es una matriz triangular inferior con unos en la diagonal principal y U es una matriz triangular superior. El proceso para encontrar estas matrices es:
La matriz U se obtiene como resultado de hacer Gauss a la matriz A y los elementos de L son los multiplicadores excepto la diagonal principal que como hemos dicho son unos, es decir

$$\begin{cases} l_{kk} = 1 \text{ para cada } k = 1, ..., n \\ \\ l_{ij} = \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}} \text{ para cada } j = 1, ..., n-1 \text{ y para cada } j = i+1, ..., n \end{cases}$$

El resto de elementos son ceros.

Teorema 4.
Dicha descomposición existe si $a_{jj}^{(j)}$ (para $1 \leq j \leq n$) es más grande que una cierta tolerancia. Si además la matriz A es regular, dicha descomposición es única.

Con esta descomposición, si se quiere resolver $A \cdot x{=}b$, lo que se hace es resolver primero $L \cdot y{=}b$ y una vez encontrado el vector y se resuelve $U \cdot x{=}y$ que son sencillos pues tenemos L y U matrices triangulares y por tanto se aplicaría el método de sustitución hacia adelante y hacia atrás respectivamente. La ventaja de esta descomposición respecto a Gauss, es que si se quiere modificar el vector b no hay que repetir el proceso desde el principio, sino simplemente aplicar de nuevo el método de sustitución hacia adelante y hacia atrás cambiando los elementos de b que se han modificado (por lo que el vector y también se verá afectado).

2.1.5. Descomposición LU con pivotaje

Es una variante del método anterior y permite solventar la condición necesaria que tenía que satisfacer el método anterior para que la descomposición existiera.

Teorema 5.
Si A es una matriz cuadrada y regular entonces existe una matriz de permutación P tal que $P \cdot A$ admita una descomposición LU. Además, fijada la matriz P, las matrices L y U son únicas.

En este método, la matriz L se obtiene de la misma forma que el método anterior pero la matriz U se obtiene haciendo Gauss con pivotaje, necesitando matrices de permutaciones. Una matriz de permutación es una matriz P tal que el producto $P \cdot A$ intercambia las filas de la matriz A. Así, si queremos intercambiar la fila i por la fila j de A haremos $P_{ij} \cdot A$, donde P_{ij} es la matriz de permutación que se obtiene

como la matriz identidad intercambiando la fila i por la fila j. De esta manera si P es el producto de todas estas matrices de permutación se resuelve $L \cdot y = P \cdot b$ (de hecho $P \cdot b$ se obtiene directamente, si cada vez que intercambiamos la fila i por la fila j de A tambien intercambiamos la componente i (b_i) y la componente j (b_j) de b) y una vez encontrado el vector y se resuelve $U \cdot x = y$.

Observación 5.
Si A es una matriz banda (p, q) entonces L es una matriz $(p, 1)$ y U es una matriz $(1, q)$

Observación 6.
Tenemos que:

$$A \cdot x = b \iff L \cdot U \cdot x = b \iff U \cdot x = L^{-1} \cdot b$$

Esta última igualdad es el método de Gauss pero en vez de encontrar la matriz L correspondiente a la descomposición LU, te modifica directamente el vector de términos independientes b. Es por ello que el método de Gauss es equivalente a una descomposición LU
Puede haber casos en que $\kappa(L)$ y/o $\kappa(U)$ empeore a $\kappa(A)$ por lo cual es posible que la matriz A esté bien condicionada pero que al hacer la descomposición LU (o Gauss), las matrices no lo estén. Es por ello que estos métodos no son muy eficientes para dichos casos.

2.1.6. Variantes de la descomposición LU

Hay algunos métodos que persiguen el mismo objetivo que la descomposición LU sin necesidad de hacer eliminación gaussiana bajo las mismas condiciones. Aquí veremos dos métodos:

1. **Método de Doolittle:**
   El algoritmo que permite encontrar los elementos de las matrices L y U son:

$$\begin{cases} u_{kj} = a_{kj} - \sum_{r=1}^{k-1} l_{kr} \cdot u_{rj} \text{ para } j = k, \cdots, n \\ \\ l_{kk} = 1 \\ \\ l_{ik} = \frac{a_{ik} - \sum_{r=1}^{k-1} l_{ir} \cdot u_{rk}}{u_{kk}} \text{ para } i = k+1, \cdots, n \end{cases}$$

   para $1 \leq k \leq n$.
   Para que u_{kk} $(k = 1, \cdots, n)$ sea no nulo es necesario que $\det A_k \neq 0$ para $1 \leq k \leq n$. El orden que se ha de calcular es muy importante y es el siguiente: u_{11}, u_{12}, $\cdots$, u_{1n}, l_{11}, l_{21}, l_{31}, $\cdots$, l_{n1}, u_{22}, u_{23}, $\cdots$, u_{2n}, l_{22}, l_{32}, l_{42}, $\cdots$, l_{n2}, $\cdots$, u_{n-1n-1}, u_{n-1n}, l_{nn-1} y u_{nn}

2. **Método de Crout:**
   Este método es similar al anterior pero ahora es la matriz U la que posee unos

en la diagonal principal. El algoritmo que permite encontrar los elementos de las matrices L y U son:

$$\begin{cases} l_{ik} = a_{ik} - \sum_{r=1}^{k-1} l_{ir} \cdot u_{rk} \text{ para } j = k, \cdots, n \\ \\ u_{kk} = 1 \\ \\ l_{kj} = \frac{a_{kj} - \sum_{r=1}^{k-1} l_{kr} \cdot u_{rj}}{l_{kk}} \text{ para } i = k+1, \cdots, n \end{cases}$$

para $1 \leq k \leq n$.
Para que l_{kk} $(k = 1, \cdots, n)$ sea no nulo es necesario que $\det A_k \neq 0$ para $1 \leq k \leq n$.
El orden que se ha de calcular es muy importante y es el siguiente: l_{11}, l_{12}, $\cdots$, l_{1n}, u_{11}, u_{21}, u_{31}, $\cdots$, u_{n1}, l_{22}, l_{23}, $\cdots$, l_{2n}, u_{22}, u_{32}, u_{42}, $\cdots$, u_{n2}, $\cdots$, l_{n-1n-1}, l_{n-1n}, u_{nn-1} y l_{nn}

2.1.7. Descomposición Cholesky

Para matrices simétricas es mejor realizar la factorización Cholesky. Dicha factorización parte de la descomposición $A = L \cdot D \cdot L^t$ donde L es una matriz triangular inferior con unos en la diagonal y D es una matriz diagonal. Estas matrices se obtienen como:

$$\begin{cases} d_{kk} = a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2 \cdot d_{jj} \\ \\ l_{kk} = 1 \\ \\ l_{ik} = \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot d_{jj} \cdot l_{kj}}{d_{kk}} \text{ para } i = k+1, \cdots, n \end{cases}$$

para $k = 1, 2, \cdots, n$. Dicha factorización existe si d_{kk} no es cero o un número muy pequeño porque entonces para calcular el l_{ik} se estaría dividiendo prácticamente entre cero y daría problemas. El resto de operaciones (suma y resta) no dan ningún error.
Antes de continuar necesitaremos el criterio de Sylvester:

Teorema 6. *(Criterio de Sylvester)*
Una matriz simétrica es definida positiva $\iff$ todos los menores tienen determinante positivo.
Si la matriz es diagonal o triangular, esta última condición equivale a que los elementos de la diagonal principal sean todos positivos

De esta manera, si A es una matriz definida positiva, la matriz D también lo es y por tanto $d_{ii} > 0$ $1 \leq i \leq n$. De esta manera la matriz

$$D^{\frac{1}{2}} = \begin{pmatrix} \sqrt{d_{11}} & 0 & 0 & \cdot & 0 \\ 0 & \sqrt{d_{22}} & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{d_{33}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{d_{nn}} \end{pmatrix}$$

esta bien definida y se puede ver que satisface $D = D^{\frac{1}{2}} \cdot D^{\frac{1}{2}}$. Así, si definmos $\mathcal{L}$ como la matriz $L \cdot D^{\frac{1}{2}}$, A puede expresarse en la forma $A = \mathcal{L} \cdot \mathcal{L}^t$ llamada factorización de Cholesky. Se puede observar que $\mathcal{L}$ es una matriz triangular superior.

Teorema 7.
Dicha factorización, con elementos diagonales de $\mathcal{L}$ positivos, existe y es única si y sólo si A es simétrica y definida positiva.

2.1.8. Descomposición QR

El objetivo de este método es descomponer la matriz A como el producto de dos matrices Q y R, donde Q es una matriz ortogonal y R es una matriz triangular superior. Para toda matriz real y cuadrada A, existe su descomposición QR. Si A es una matriz regular tenemos que dicha descomposición es única. Con esta descomposición, si se quiere resolver $A \cdot x{=}b$ lo que se hace es resolver $R \cdot x{=}y$ donde $y{=}Q^t \cdot b$. Hay varios tipos de formas de encontrar estas matrices Q y R. Nosotros lo veremos de dos formas.

- **QR Householder:** La primera es usar transformaciones de Householder. Para cada vector no nulo $u \in \Re^n$, definimos la matriz de Householder por:

$$P(u) = Id - \frac{2}{u^t \cdot u} \cdot u \cdot u^t$$

De este modo, eligiendo el vector u adecuadamente se puede obtener la matriz $P(u)$ de tal modo que cuando multipliquemos por un vector v obtengamos el primer elemento no nulo y el resto de elementos cero. En nuestro caso partiendo de A se puede encontrar una matriz de Householder $P(u)$ tal que $A_1{=}P(u) \cdot A$ tenga como primera columna el primer elemento diferente de cero y el resto con ceros. Repitiendo el paso con A_1 se puede encontrar una matriz

$$Q_1 \begin{pmatrix} 1 & 0 \\ 0 & P(u_1) \end{pmatrix}$$

tal que $A_2{=}Q_1 \cdot A_1$ tenga la primera columna idéntica a A_1 y la segunda con los dos primeros elementos diferente de cero y el resto con ceros. Repitiendo el paso con A_2 se puede encontrar una matriz

$$Q_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & P(u_2) \end{pmatrix}$$

tal que $A_3{=}Q_2 \cdot A_2$ tenga la dos primeras columnas identicas a A_2 y la tercera con los tres primeros elementos diferente de cero y el resto con ceros. Repitiendo el paso k veces, se puede encontrar una matriz

$$Q_k = \begin{pmatrix} Id_k & 0 \\ 0 & P(u_k) \end{pmatrix},$$

donde Id_k es la matriz identidad de dimensión $k \times k$ y $P(u_k)$ sea una matriz de Householder, tal que $Q_k \cdot A_{k-1}$ mantenga las primeras k columnas de A_{k-1} y modifique la columna k-ésima por un vector cuyos primeros k elementos seano nulo y el resto con cero. Repitiendo el paso hasta $k = n - 1$ obtenemos una matriz triangular.
Con esta descomposición, si se quiere resolver $A \cdot x = b$ lo que se hace es resolver $R \cdot x = y$ donde $y = Q^t \cdot b$.

- **QR Givens:** La segunda es usar rotaciones de Givens. Si notamos c como el $cos(\theta)$ y s como el $sin(\theta)$, una rotación de Givens es:

$$R(i,j,\theta) = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \end{array}$$

El objetivo es encontrar los valores de c y de s tal que mediante las rotaciones de Givens hagan 0 los elementos por debajo de la diagonal principal para conseguir una matriz triangular superior. Para ello, hemos de igualar:

$$\begin{pmatrix} c & s \\ s & c \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2} \\ 0 \end{pmatrix}$$

Así $c = \frac{x}{\sqrt{x^2+y^2}}$ y $s = \frac{y}{\sqrt{x^2+y^2}}$. Es por ello que si la matriz es

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

el orden en el que se anula es a_{n1} (llamemosle G_1 a esta matriz de Givens), a_{n-11} (llamemos G_2 a esta matriz), $\cdots$, a_{21} (llamemos G_{n-1} a esta matriz), a_{n2} (llamemos G_n a esta matriz), etc. Así, $R = \cdots G_n \cdot G_{n-1} \cdots G_2 \cdot G_1 \cdot A$ y $Q = \cdots G_n \cdot G_{n-1} \cdots G_2 \cdot G_1$ que es ortogonal pues cada matriz de rotación de Givens lo es y el producto de ortogonales es una matriz ortogonal.

Observación 7.
Una de las ventajas de la descomposición QR es que $\kappa(A) = \kappa(R)$ (ya que $||Q||_2 = 1$) por lo que si la matriz A está bien definido la matriz R también lo está. Con esto, concluimos que el método no va a empeorar.

Observación 8.
Como se puede apreciar, el método de Givens va haciendo cero elemento a elemento de la columna, mientras que el de Householder los hace todo al mismo tiempo. Es por ello, que habitualmente es lógico usar Householder antes que Givens. Sin embargo, hay casos (por ejemplo, cuando la matriz A es escasa) en que Givens es preferible.

2.2. Métodos iterativos

Un método iterativo es un método por el que se construye una sucesión de forma que si su límite existe, este sea la solución del sistema que queremos resolver. En teoría dificultaría el hecho de darnos la solución real ya que requeriría un número infinito de pasos y en algún punto habría que parar.
Estos métodos, consisten en encontrar una matriz B y un vector c tal que transformemos el problema de hallar la solución de $A \cdot x = b$ en hallar el límite de la sucesión definida por $x^{(k+1)} = B \cdot x^{(k)} + c$ equivalente al primero.

Teorema 8.
Supongamos que B diagonaliza. Entonces:
El método iterativo $x^{k+1} = B \cdot x^k + c$ converge independientemente la condición inicial $\iff \rho(B) < 1$

Notaremos la matriz L como la matriz que tiene por elementos

$$l_{ij} = \begin{cases} a_{ij} & : j < i \\ 0 & : j \geq i \end{cases},$$

la matriz U como la matriz que tiene por elementos

$$u_{ij} = \begin{cases} a_{ij} & : i < j \\ 0 & : i \geq j \end{cases},$$

la matriz D como la matriz que tiene por elementos

$$d_{ij} = \begin{cases} a_{ij} & : i = j \\ 0 & : i \neq j \end{cases}$$

Así, la matriz D^{-1} es la matriz que tiene por elementos

$$d_{ij}^{-1} = \begin{cases} \frac{1}{a_{ij}} & : i = j \\ 0 & : i \neq j \end{cases}.$$

De tal modo, $A{=}L + D + U$.

2.2.1. Jacobi

La iteración de Jacobi se basa en:

$$A \cdot x = b \iff (L + D + U) \cdot x = b \iff (L + U) \cdot x + D \cdot x = b \iff$$

$D \cdot x = b - (L+U) \cdot x \iff x = D^{-1} \cdot (b - (L+U) \cdot x) \iff x = -D^{-1} \cdot (L+U) \cdot x + D^{-1} \cdot b$

Por tanto, se puede definir la siguiente recurrencia:

$$\boxed{x^{(k+1)} = -D^{-1} \cdot (L + U) \cdot x^{(k)} + D^{-1} \cdot b, \forall k \geq 1}$$

Así $B = -D^{-1} \cdot (L + U)$ y $c = D^{-1} \cdot b$. Expresado en componentes:

$$\boxed{x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} \cdot x_j^{(k)}}{a_{ii}}, \forall k \geq 1,\ 1 \leq i \leq n}$$

2.2.2. Gauss-Seidel

La iteración de Gauss-Seidel se basa en:

$A \cdot x = b \iff (L+D+U) \cdot x = b \iff L \cdot x + D \cdot x + U \cdot x = b \iff D \cdot x = b - L \cdot x - U \cdot x$

Por tanto,

$$D \cdot x^{(k+1)} = b - L \cdot x^{(k+1)} - U \cdot x^{(k)} \iff (D + L) \cdot x^{(k+1)} = b - U \cdot x^{(k)} \iff$$

$$\iff x^{(k+1)} = (D + L)^{-1} \cdot (b - U \cdot x^{(k)})$$

Finalmente, aislando, se puede definir la recurrencia:

$$\boxed{x^{(k+1)} = -(D + L)^{-1} \cdot U \cdot x^{(k)} + (D + L)^{-1} \cdot b, \forall k \geq 1}$$

Así $B = -(D + L)^{-1} \cdot U$ y $c = (D + L)^{-1} \cdot b$. Expresado en componentes:

$$\boxed{x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} \cdot x_j^{(k+1)} - \sum_{j>i} a_{ij} \cdot x_j^{(k)}}{a_{ii}}, \forall k \geq 1,\ 1 \leq i \leq n}$$

NOTA: Hay que decir que para que D y $D + L$ tengan inversa es preciso que las matrices D y $D + L$ sean regulares, es decir que sus determinantes sean diferente de 0. El determinante de ambas es el producto de los elementos de la diagonal (por ser triangular) y puesto que los elementos de la diagonal es el mismo concluimos que el determinante tambien es el mismo. Así, es necesario que $det(D) = det(D + L) = \prod_{i=1}^{n} a_{ii} \neq 0$, es decir que $a_{ii} \neq 0$ para todo $i = 1, ..., n$. que en estos dos ejemplos es obvio que lo cumple por el criterio de Sylvester.

2.2.3. Sobrerelajación (SOR)

Sea un $w \in \Re \setminus 0$. Así:

$$A \cdot x = b \iff w \cdot A \cdot x = w \cdot b \iff w \cdot (L + D + U) \cdot x = w \cdot b \iff$$

$$\iff w \cdot (L+D+U) \cdot x + D \cdot x - D \cdot x = w \cdot b \iff D \cdot x = D \cdot x + w(\cdot b - L \cdot x - (D+U) \cdot x)$$

Por tanto,

$$D \cdot x^{(k+1)} = D \cdot x^{(k)} + w \cdot (b - L \cdot x^{(k+1)} - (D+U) \cdot x^{(k)}) \iff$$

$$\iff x^{(k+1)} = x^{(k)} + w \cdot D^{-1} \cdot (b - L \cdot x^{(k+1)} - (D+U) \cdot x^{(k)})$$

Arreglándolo queda:

$$(Id + w \cdot D^{-1} \cdot L) \cdot x^{(k+1)} = (Id - w \cdot D^{-1} \cdot (D+U)) \cdot x^{(k)} + w \cdot D^{-1} \cdot b$$

La matriz $M{=}Id + w \cdot D^{-1} \cdot L$ tiene por elementos

$$m_{ij} = \begin{cases} w \cdot a_{ij}/a_{ii} & : j < i \\ 0 & : j > i \\ 1 & : j = i \end{cases},$$

de tal manera que su determinante es el producto de los elementos de la diagonal (por ser una matriz triangular inferior), es decir

$$det(Id + w \cdot D^{-1} \cdot L) = \prod_{i=1}^{n} m_{ii} = \prod_{i=1}^{n} 1 = 1 \neq 0$$

Asi pues existe la inversa de $Id{+}w{\cdot}D^{-1}{\cdot}L$ y aislando se puede definir la recurrencia:

$$\boxed{x^{(k+1)} = (Id + w \cdot D^{-1} \cdot L)^{-1} \cdot [(1-w) \cdot Id - w \cdot D^{-1} \cdot U] \cdot x^{(k)} +}$$

$$\boxed{+w \cdot (Id + w \cdot D^{-1} \cdot L)^{-1} \cdot D^{-1} \cdot b, \, \forall k \geq 1}$$

Así $B = (Id{+}w{\cdot}D^{-1}{\cdot}L)^{-1}{\cdot}[(1{-}w){\cdot}Id{-}w{\cdot}D^{-1}{\cdot}U]$ y $c = w{\cdot}(Id{+}w{\cdot}D^{-1}{\cdot}L)^{-1}{\cdot}D^{-1}{\cdot}b$. Expresado en componentes:

$$\boxed{x_i^{(k+1)} = x_i^{(k)} + w \cdot \frac{b_i - \sum_{j<i} a_{ij} \cdot x_j^{(k+1)} - \sum_{j \geq i} a_{ij} \cdot x_j^{(k)}}{a_{ii}}, \, \forall k \geq 1, \, \forall 1 \leq i \leq n}$$

NOTA 1: $\rho(B_{SOR}) \geq |w - 1|$
NOTA 2: Observemos que si $w = 1$, el método de SOR es el mismo que el método de Gauss-Seidel.
NOTA 3: ¿Cuál es el valor óptimo del w? La respuesta es que en general no lo sabemos. Sin embargo, con nuestros dos ejemplos, podemos restringir un poco los valores del w para que cualquier real no nos sirva. Para ello es necesario enunciar el siguiente teorema:

Teorema 9. *(Ostranski-Beich)*
Si queremos resolver $A \cdot x = b$ con A simétrica y definida positiva, entonces SOR converge $\forall w \in (0, 2)$

Teorema 10.
Si A es una matriz simétrica, definida positiva, real y tridiagonal a bloques entonces $\rho(B_{GS}) = \rho(B_J)^2$, el w óptimo de SOR es:

$$\bar{w}\frac{2}{1+\sqrt{1-\rho(B_{GS})}}$$

y $\rho(B_{SOR}) = \bar{w} - 1$

2.3. Métodos de Krylov

Un subespacio de Krylov de orden r generado por una matriz cuadrada A de orden n y un vector v es el subespacio vectorial generado por $A^k \cdot v$ con $k < r$, es decir

$$\mathcal{K}_r(A, v) = \text{span}\left\{v, Av, \ldots, A^{r-1}v\right\}$$

Se utilizan, mediant métodos iterativos, para el calculo de valores propios, vectores propios o sobretodo para resolver sistemas de ecuaciones lineales con matrices dispersas. Es por ello que los algoritmos que usan este subespacio se les conoce por métodos del subespacio de Krylov. En este apartado veremos algunos sobre la resolución de sistemas de ecuaciones (en el apartado de valores propios veremos otros sobre el cálculo de dichos valores).

2.3.1. Minimización

Este método se puede aplicar para resolver $A \cdot x = b$ cuando A sea una matriz simétrica y definida positiva. Partimos de la siguiente función

$$H(x) = \frac{1}{2} \cdot x^t \cdot A \cdot x - x^t \cdot b$$

Su gradiente es

$$\nabla H(x) = A \cdot x - b$$

y su matriz hessiana es A. Como estamos en el caso que A es definida positiva podemos concluir que la solución de $\nabla H(x) = 0$ es el mínimo de H; es decir, que encontrar la solución de $A \cdot x = b$ es equivalente a encontrar el mínimo de H. Por ello, este método es un método que encuentra este mínimo que buscamos. La idea es avanzar en la dirección de máxima pendiente que será $p_k = \nabla H(x)$. Así pues, definimos el sistema iterativo:

$$x^{(k+1)} = x^{(k)} - \alpha_k \cdot p_k$$

Buscando el α_k mediante un problema de minimización de la función $H(x_k - \alpha_k \cdot p_k)$ obtenemos que:

$$\alpha_k = \frac{p_k^t \cdot p_k}{p_k^t \cdot A \cdot p_k}$$

Substituyendo los valores hallados donde correpondan obtenermos el sistema iterativo siguiente:
Dado un $x^{(0)}$, definimos $p_0 = r_0 = b - A \cdot x^{(0)}$ y

$$\begin{cases} x^{(k+1)} = x^{(k)} + \frac{p_k^t \cdot p_k}{p_k^t \cdot A \cdot p_k} \cdot p_k \\ p_{k+1} = A \cdot x^{(k)} - b \end{cases}$$

$\forall k \geq 1$

2.3.2. Gradiente conjugado

Este método es una mejora del método anterior. Definimos como antes el sistema iterativo:

$$x^{(k+1)} = x^{(k)} + \alpha_k \cdot p_k$$

con $p_0 = r_0 = b - A \cdot x^{(0)}$, $r_{k+1} = r_k + \alpha_k \cdot A \cdot p_k$ y $p_{k+1} = r_k + \beta_k \cdot p_k$.
Si exigimos que las direcciones p_k sean A-conjugada, es decir que $p_k^t \cdot A \cdot p_k$ sea 0, obtenemos:

$$\beta_k = \frac{r_k^t \cdot A \cdot p_{k-1}}{p_{k-1}^t \cdot A \cdot p_{k-1}} = \frac{r_{k+1}^t \cdot r_{k+1}}{r_k^t \cdot r_k}$$

y buscando el minimo de $H(x_k + \alpha_k \cdot p_k)$ obtenemos:

$$\alpha_k = \frac{r_k^t \cdot p_k}{p_{k-1}^t \cdot A \cdot p_k} = \frac{r_k^t \cdot r_k}{r_k^t \cdot A \cdot p_k}$$

Substituyendo los valores hallados donde correpondan obtenermos el sistema iterativo siguiente:

Dado un $x^{(0)}$, definimos $p_0 = r_0 = b - A \cdot x^{(0)}$ y

$$\begin{cases} x^{(k+1)} = x^{(k)} + \frac{r_k^t \cdot r_k}{r_k^t \cdot A \cdot p_k} \cdot p_k \\ r_{k+1} = r_k + \frac{r_k^t \cdot r_k}{r_k^t \cdot A \cdot p_k} \cdot A \cdot p_k \\ p_{k+1} = r_k + \frac{r_{k+1}^t \cdot r_{k+1}}{r_k^t \cdot r_k} \cdot p_k \end{cases}$$

$\forall k \geq 0$

2.3.3. Gradiente biconjugado

Se basa en el método anterior pero tenemos un pseudo-gradiente s_k y una pseudo-dirección q_k. Definimos como antes el sistema iterativo:

$$x^{(k+1)} = x^{(k)} + \alpha_k \cdot p_k$$

con $p_0 = r_0 = q_0 = s_0 = b - A \cdot x^0$, $r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k$ $s_{k+1} = s_k - \alpha_k \cdot A^t \cdot q_k$, $p_{k+1} = r_k + \beta_k \cdot p_k$ y $q_{k+1} = s_k + \beta_k \cdot q_k$. El método se construye de tal forma que los pseudo-gradientes s_k sean ortogonales a los gradientes r_k y que las pseudo-direcciones de descenso p_k sean A-ortogonales a las direcciones de descenso q_k. Así obtenemos:

$$\alpha_k = \frac{s_k^t \cdot r_k}{q_k^t \cdot A \cdot p_k}$$

$$\beta_k = \frac{s_{k+1}^t \cdot r_{k+1}}{s_k^t \cdot r_k}$$

Substituyendo los valores hallados donde correpondan obtenermos el sistema iterativo siguiente:

Dado un $x^{(0)}$, definimos $p_0 = r_0 = q_0 = s_0 = b - A \cdot x^{(0)}$ y

$$\begin{cases} x^{(k+1)} = x^{(k)} + \frac{s_k^t \cdot r_k}{q_k^t \cdot A \cdot p_k} \cdot p_k \\ r_{k+1} = r_k - \frac{s_k^t \cdot r_k}{q_k^t \cdot A \cdot p_k} \cdot A \cdot p_k \\ s_{k+1} = s_k - \frac{s_k^t \cdot r_k}{q_k^t \cdot A \cdot p_k} \cdot A^t \cdot q_k \\ p_{k+1} = r_k + \frac{s_{k+1}^t \cdot r_{k+1}}{s_k^t \cdot r_k} \cdot p_k \\ q_{k+1} = r_k + \frac{s_{k+1}^t \cdot r_{k+1}}{s_k^t \cdot r_k} \cdot q_k \end{cases}$$

$\forall k \geq 0$

2.4. Error en la resolución y comparación de los métodos

Para ver la eficacia de estos métodos los aplicaremos a varios ejemplos. Como ya hemos comentado, en los casos que conocemos la solución, si x es la solución que nos da el método empleado y $\bar{x}$ es la solución real calcularemos la norma del vector error definido por $e_r = x - \bar{x}$, que queremos que sea lo más pequeña posible. En los sistemas de ecuaciones aleatorios no podemos calcular la norma del vector error, lo que haremos pues será calcular la norma del vector residual definido por $r = A \cdot x - b$. Como antes, nuestro objetivo es que la norma de este vector residual sea lo más pequeña posible.
El primer ejemplo concreto que estudiaremos será un sistema cuya matriz es la de

Hilbert (es decir la matriz formada por los $a_{ij} = \frac{1}{i+j-1}$, con $i, j = 1, ..., n$) y vector de términos independientes la suma de las filas, es decir

$$b = \begin{pmatrix} \sum_{i=1}^{n} a_{1i} \\ \vdots \\ \sum_{i=1}^{n} a_{ni} \end{pmatrix}.$$

Este ejemplo se ha elegido porque es sencillo generar la matriz del sistema, cuya solución es:

$$x = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^t.$$

Este ejemplo tiene el inconveniente que dicha matriz está mal condicionada (tal y como se dijo en los preliminares, una pequeña perturbación en la matriz, afecta considerablemente a la solución, alejándola de la real). También se suele decir que está mal condicionada, si en algún momento se divide entre cero o un número muy pequeño; o si en algún paso del proceso el redondeo aumenta significativamente y estos cambios de décimas se van repitiendo a lo largo del proceso hasta tener una gran diferencia. Por ejemplo,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

tiene como solución $(1, 1)$, en cambio,

$$\begin{pmatrix} 10^{-10} & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

por Gauss con una precisión del 10^{-12} nos da como solución $(0, 1)$ que obviamente no es solución pero que el sistema prácticamente es el mismo ya que 10^{-10} es un número muy pequeño y se podía tratar como 0. Es por ello, que este ejemplo, nos servirá para probar los métodos de manera eficiente.
El segundo ejemplo que estudiaremos consistirá en resolver la ecuación $\Delta u = f(x, y)$ definida para $(x, y) \in Q$, donde u: $Q \longrightarrow \Re$ es una función, Δu es el laplaciano de u ($\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$) y Q es el cuadrado unidad $[0, 1] \times [0, 1]$, Se desea calcular $u = u(x, y)$ de la que se conocen los valores en el borde de Q. Para ello, cogemos una malla de n puntos equidistantes tanto en x como de y (llamada malla equiespaciada de n puntos). Definimos $h = \frac{1}{n+1}$, $x_i = \frac{i}{n+1}$, $y_j = \frac{j}{n+1}$, $u_{i,j}$=$u(x_i, x_j)$ y $f_{i,j}$=$f(x_i, x_j)$ para $0 \leq i, j \leq n + 1$. Así los valores $u_{0,j}$, $u_{n+1,j}$, $u_{i,0}$ y $u_{i,n+1}$ para $1 \leq i, j \leq n$ son conocidos. El objetivo es encontrar los valores de $u_{i,j}$ para $1 \leq i, j \leq n$. Usando la fórmula centrada de la segunda derivada, es decir $\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2}$ y $\frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{h^2}$, obtengo:

$$\frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{h^2} = f_{i,j}$$

O equivalentemente:

$$u_{i+1,j} + u_{i-1,j} - 4 \cdot u_{i,j} + u_{i,j+1} + u_{i,j-1} = h^2 \cdot f_{i,j}$$

Teniendo en cuenta que $u_{0,j}$, $u_{n+1,j}$, $u_{i,0}$ y $u_{i,n+1}$ son conocidos nos queda el sistema de dimensión, $n^2 \times n^2$, $A \cdot \mathfrak{u} = b$:

$$\begin{pmatrix} T_n & Id_n & 0 & 0 & ... & 0 \\ Id_n & T_n & Id_n & 0 & ... & 0 \\ 0 & Id_n & T_n & Id_n & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & ... & & \ddots & \ddots & Id_n \\ 0 & ... & 0 & 0 & Id_n & T_n \end{pmatrix} \cdot \begin{pmatrix} u_{11} \\ u_{12} \\ \vdots \\ u_{1n} \\ u_{21} \\ \vdots \\ u_{nn} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \\ b_{n+1} \\ \vdots \\ b_{n^2} \end{pmatrix}$$

donde T_n es la matriz tridiagonal $n \times n$

$$T_n = \begin{pmatrix} -4 & 1 & 0 & 0 & ... & 0 \\ 1 & -4 & 1 & 0 & ... & 0 \\ 0 & 1 & -4 & 1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & ... & & \ddots & \ddots & 1 \\ 0 & ... & 0 & 0 & 1 & -4 \end{pmatrix},$$

Id_n es la matriz identidad $n \times n$ y $b_i = h^2 \cdot f_{i,j} - \frac{1}{h^2} \cdot uf$ donde uf indica los valores de u en la frontera que por hipotesis es conocido (por ejemplo para $i = 1$, uf sería $u_{0,1} + u_{1,0}$). Asi la matriz A se podria compactar como:

$$\begin{cases} a_{k,k} = -4 \\ a_{k,k+1} = 1 \text{ si } k + 1 \leq n^2 \\ a_{k,k-1} = 1 \text{ si } k - 1 > 0 \\ a_{k,k+n} = 1 \text{ si } k + n \leq n^2 \\ a_{k,k-n} = 1 \text{ si } k - n > 0 \end{cases}$$

para $1 \leq k \leq n^2$. Este ejemplo es útil porque de entrada el método empleado tiene un error (es decir si multiplicamos $A \cdot \bar{\mathfrak{u}} := \bar{b}$, donde $\bar{\mathfrak{u}}$ indica la solución real del sistema, entonces $\bar{b}$ y b son distintos ya que el método empleado es aproximado y no igual) por lo que prolonga el error de la solución pero es un buen ejemplo por ser una matriz escasa, simétrica, definida positiva y tridiagonal por bloques donde sus bloques son matrices tridiagonales (T_n) o matrices identidades (Id_n). La solución de dicho sistema se sabe pues previamente se elije una funcion u para calcular su laplaciano f y así calcular el vector b correspondiente.

Por último, fijaremos la dimensión del sistema por 100 y resolveremos 10 sistemas de ecuaciones aleatorios con número de condición fijado, κ. Para ello, primero crearemos una matriz M y un vector b aleatoriamente. A continuación, definiremos A por

$$A = Q \cdot D \cdot Q^t$$

donde Q es la matriz que se obtiene haciendo la descomposición QR a la matriz M y D es una matriz diagonal aleatoria de la forma:

$$D = \begin{pmatrix} \lambda_1 & 0 & 0 & \cdot & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

con $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n$ tal que $\kappa = \left|\frac{\lambda_1}{\lambda_n}\right|$. Y finalmente, resolveremos $A \cdot x = b$ con $\kappa_2(A) = \kappa$ (ya que $||A||_2 = ||Q \cdot D \cdot Q^t||_2 = ||D||_2$ puesto que $||Q||_2 = ||Q^t||_2 = 1$ por ser Q ortogonal).
En todos los ejemplos, fijaremos la tolerancia por 10^{-12} y para los sistemas iterativos permitiremos un máximo de 1000 iteraciones con condición inicial x donde, si no se especifica lo contrario, x es el vector formado por los elementos

$$x_i = \begin{cases} 1 - \frac{1}{i+5} & \text{si } i \quad \text{es par} \\ 1 & \text{si } i \quad \text{es impar} \end{cases}$$

para el primer ejemplo y $x_i = 0$ para el resto.

- Gauss:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	7	$1{,}790613418916703 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$9{,}407581464093986 \cdot 10^{-07}$
3	$1{,}611621708230620 \cdot 10^{-14}$	9	$3{,}553701971479750 \cdot 10^{-05}$
4	$7{,}182656651154077 \cdot 10^{-13}$	10	$6{,}198350689794145 \cdot 10^{-04}$
5	$6{,}117974111008171 \cdot 10^{-13}$	11	$1{,}717259534932144 \cdot 10^{-02}$
6	$5{,}820214336381012 \cdot 10^{-10}$	12	$2{,}272195371689464 \cdot 10^{-02}$

Tabla 1: Error del primer ejemplo usando el método de Gauss con condición de parada

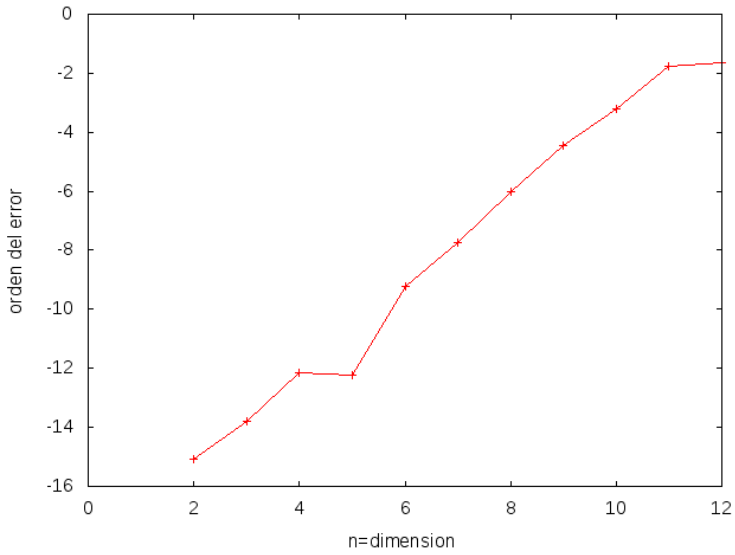


Figura 1: Error del primer ejemplo en escala logarítmica usando el método de Gauss

Tal y como se puede observar, el método funciona hasta dimensión 12, donde a partir de allí el proceso finaliza por tener algún pivote que sea más pequeño que nuestra tolerancia. A dichos pivotes les llamaremos, a partir de ahora, pivotes problemáticos. De entrada, parece que el método funciona bien aunque cada vez el error va perdiendo precisión, considerándose buenos datos exceptuando para dimensión 10, 11 y 12 donde se pueden considerar aceptables (pues según el contexto, el orden de 10^{-4} o 10^{-2} puede ser un error demasiado grande númericamente). Arreglando el programa para que no pare en los pivotes problemáticos, se puede observar la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
13	6,337226713819835	70	$2{,}385907395567054 \cdot 10^{+02}$
20	$2{,}047768457132124 \cdot 10^{+03}$	80	$8{,}388321923261125 \cdot 10^{+03}$
30	$2{,}533035013345059 \cdot 10$	90	$2{,}276382283581402 \cdot 10^{+03}$
40	$1{,}473363784014535 \cdot 10^{+02}$	100	$2{,}773849398939395 \cdot 10^{+04}$
50	$5{,}739487953861078 \cdot 10^{+02}$	1000	$1{,}251758309948761 \cdot 10^{+06}$
60	$8{,}761744979209279 \cdot 10^{+02}$		

Tabla 2: Error del primer ejemplo usando el método de Gauss sin usar pivotaje

Estos grandes errores provienen por el hecho que conforme aumenta la dimensión, los pivotes empiezan a ser más pequeños por lo que al dividirlos conlleva un error que se va acumulando y aumentando significativamente conforme aumenta la dimensión. El primer pivote problemático proviene del $a_{11,11}^{(11)} = 9{,}132062023240329 \cdot 10^{-14}$ para cualquier dimensión superior a 13, llegando a alcanzar 538 pivotes problemáticos para $n = 1000$.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202391027821 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293663965129 \cdot 10^{-04}$
9	$2{,}158044791559693 \cdot 10^{-03}$	1600	$2{,}208714582493938 \cdot 10^{-04}$
16	$1{,}758522566482695 \cdot 10^{-03}$	2500	$1{,}775893530889277 \cdot 10^{-04}$
25	$1{,}479488029213543 \cdot 10^{-03}$	3600	$1{,}484882119725278 \cdot 10^{-04}$
36	$1{,}275238805384803 \cdot 10^{-03}$	4900	$1{,}275804263679398 \cdot 10^{-04}$
49	$1{,}119809922148691 \cdot 10^{-03}$	6400	$1{,}118330094669381 \cdot 10^{-04}$
64	$9{,}977841776036746 \cdot 10^{-04}$	8100	$9{,}954537040295337 \cdot 10^{-05}$
81	$8{,}995373727264779 \cdot 10^{-04}$	10000	$8{,}969010124756149 \cdot 10^{-05}$
100	$8{,}187854975044966 \cdot 10^{-04}$		

Tabla 3: Error del segundo ejemplo usando el método de Gauss

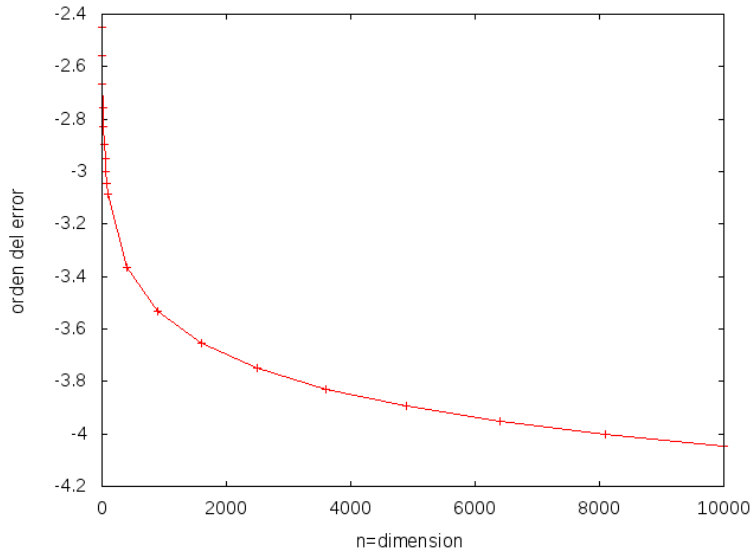


Figura 2: Error del segundo ejemplo en escala logarítmica usando el método de Gauss

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Concluimos pues que para cualquier dimensión, el método es aceptable. En contra del primer ejemplo, este método obtiene mejores resultados conforme aumenta la dimensión.

Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}133776765103534 \cdot 10^{-11}$	$1{,}000000000000586$
2	100	$3{,}178991556225995 \cdot 10^{-13}$	$8{,}889579542351672$
3	100	$3{,}466959148464888 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$5{,}586211830921018 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$3{,}227969433083943 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$4{,}421031786869020 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$3{,}822981990958598 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$3{,}474108484690415 \cdot 10^{-13}$	$7{,}673291683578189$
9	100	$3{,}672189943151701 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$4{,}263352995973894 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 4: Error de 10 sistema aleatorios usando el método de Gauss

En tal caso, se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz.

- Gauss con pivotaje:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	$0{,}000000000000000$	7	$1{,}766426493199649 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$1{,}000948450924103 \cdot 10^{-06}$
3	$1{,}406087708398790 \cdot 10^{-14}$	9	$3{,}065819020361006 \cdot 10^{-05}$
4	$6{,}665355203727891 \cdot 10^{-13}$	10	$5{,}923405102239804 \cdot 10^{-04}$
5	$1{,}033497173882049 \cdot 10^{-12}$	11	$1{,}781146100701697 \cdot 10^{-02}$
6	$5{,}385546532337181 \cdot 10^{-10}$		

Tabla 5: Error del primer ejemplo usando el método de Gauss con pivotaje con condición de parada

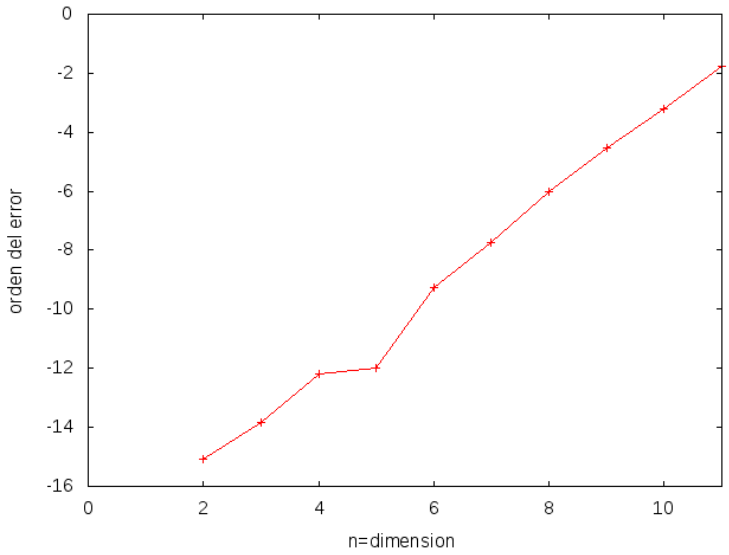


Figura 3: Error del primer ejemplo en escala logarítmica usando el método de Gauss con pivotaje

En tal caso, es hasta dimensión 11 donde el método funciona y a partir de allí hay algún pivote problemático. Como antes, parece que el método funciona bien aunque cada vez el error va perdiendo precisión, considerándose buenos datos exceptuando para dimensión 10 y 11 donde se pueden considerar aceptables. Arreglando el programa para que no pare en los pivotes problemáticos, se puede observar la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
12	5,238179647650197	70	$5{,}258980061741858 \cdot 10^{+03}$
20	$1{,}035634450029419 \cdot 10^{+03}$	80	$7{,}002133139827757 \cdot 10^{+03}$
30	$1{,}218723961620083 \cdot 10^{+04}$	90	$1{,}182761070078656 \cdot 10^{+04}$
40	$2{,}789808584155912 \cdot 10^{+03}$	100	$1{,}479977335813576 \cdot 10^{+03}$
50	$6{,}024682961980389 \cdot 10^{+03}$	1000	$1{,}323751205132458 \cdot 10^{+08}$
60	$5{,}709483255736536 \cdot 10^{+04}$		

Tabla 6: Error del primer ejemplo usando el método de Gauss con pivotaje sin condición de parada

Como antes, estos grandes errores provienen del hecho que conforme aumenta la dimensión, los pivotes empiezan a ser más pequeños por lo que al dividirlos conlleva un error que se va acumulando y aumentando significativamente conforme aumenta la dimensión. En tal caso, el primer pivote problemático depende de la dimensión del sistema y se recoge en la tabla siguiente:

n	Primer pivote problemático
12	$a_{10,10}^{(10)} = 3{,}312341821208350 \cdot 10^{-13}$
20	$a_{11,11}^{(11)} = -2{,}064298017911014 \cdot 10^{-13}$
30	$a_{17,17}^{(17)} = 6{,}705468458407020 \cdot 10^{-13}$
40	$a_{37,37}^{(37)} = 4{,}090867341368383 \cdot 10^{-13}$
50	$a_{25,25}^{(25)} = -9{,}737693063260659 \cdot 10^{-13}$
60	$a_{52,52}^{(52)} = -8{,}118077990013116 \cdot 10^{-13}$
70	$a_{49,49}^{(49)} = 9{,}574828464529725 \cdot 10^{-13}$
80	$a_{53,53}^{(53)} = -9{,}790261485754986 \cdot 10^{-13}$
90	$a_{57,57}^{(57)} = -9{,}907423670810450 \cdot 10^{-13}$
100	$a_{88,88}^{(88)} = -7{,}485389204855628 \cdot 10^{-13}$
1000	$a_{932,932}^{(932)} = -9{,}878777183751516 \cdot 10^{-13}$

Tabla 7: Primer pivote que da problemas usando el método de Gauss con pivotaje

El pivotaje, nos ha permitido prolongar, en ocasiones, la aparición del pivote problemático (por ejemplo para dimensión 1000 en el paso 11 ya nos aparecía el error mientras que ahora no es hasta el paso 932 donde aparece) aunque también ha hecho que ya en dimensión 12 obtuviésemos un pivote problemático en vez de 11 como en el método anterior (lo cual ha hecho empeorar el error para $n = 12$).
Comparándolo con el método anterior, vemos que hasta dimensión 11 el método funciona con precisión similar. No es necesario comparar para dimensión superior pues ambos métodos nos proporciona un dato erróneo al tener como pivote un número muy pequeño.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202391027821 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293728867651 \cdot 10^{-04}$
9	$2{,}158044791559693 \cdot 10^{-03}$	1600	$2{,}208714852275749 \cdot 10^{-04}$
16	$1{,}758522566482695 \cdot 10^{-03}$	2500	$1{,}775894307633910 \cdot 10^{-04}$
25	$1{,}479488029213543 \cdot 10^{-03}$	3600	$1{,}484883947749011 \cdot 10^{-04}$
36	$1{,}275238805384803 \cdot 10^{-03}$	4900	$1{,}275807954854924 \cdot 10^{-04}$
49	$1{,}119809922148691 \cdot 10^{-03}$	6400	$1{,}118336622078107 \cdot 10^{-04}$
64	$9{,}977841776036746 \cdot 10^{-04}$	8100	$9{,}954644398528988 \cdot 10^{-05}$
81	$8{,}995373727264779 \cdot 10^{-04}$	10000	$8{,}969178087310499 \cdot 10^{-05}$
100	$8{,}187854975044966 \cdot 10^{-04}$		

Tabla 8: Error del segundo ejemplo usando el método de Gauss con pivotaje

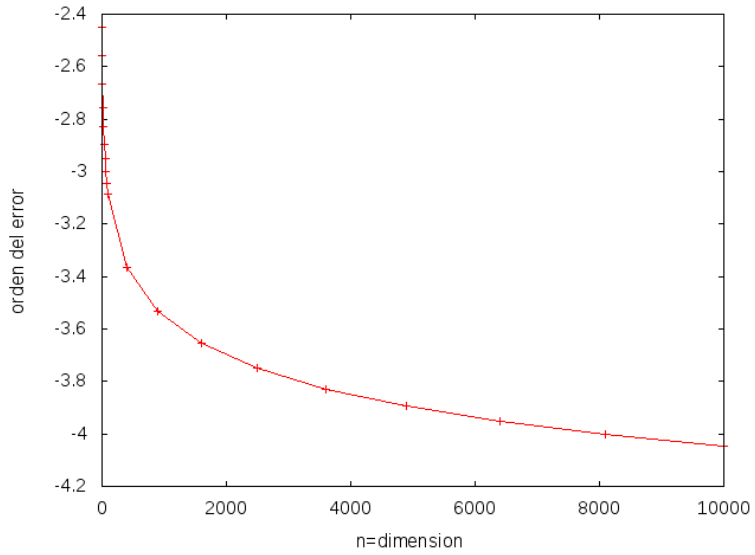


Figura 4: Error del segundo ejemplo en escala logarítmica usando el método de Gauss con pivotaje

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos observar que, en tal ejemplo, el método de Gauss con pivotaje arroja resultados similares a Gauss sin pivotaje. Como antes, dicho método nos proporciona datos aceptables para cualquier dimensión.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}133776765103534 \cdot 10^{-11}$	1,000000000000586
2	100	$3{,}178991556225995 \cdot 10^{-13}$	8,889579542351672
3	100	$3{,}466959148464888 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$5{,}586211830921018 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$3{,}227969433083943 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$4{,}421031786869020 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$3{,}822981990958598 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$3{,}474108484690415 \cdot 10^{-13}$	7,673291683578189
9	100	$3{,}672189943151701 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$4{,}263352995973894 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 9: Error de 10 sistema aleatorios usando el método de Gauss con pivotaje

En tal caso, se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con el anterior, se observa como ambos nos proporcionan datos similares.

- Gauss con pivotaje total:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	7	$1{,}792717240772115 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$9{,}072487939952670 \cdot 10^{-07}$
3	$1{,}469735699241440 \cdot 10^{-14}$	9	$2{,}818891177890915 \cdot 10^{-05}$
4	$6{,}576919217671595 \cdot 10^{-13}$	10	$5{,}623231569173975 \cdot 10^{-04}$
5	$1{,}502398485770486 \cdot 10^{-12}$	11	3,966742865464859
6	$5{,}270744600646338 \cdot 10^{-10}$		

Tabla 10: Error del primer ejemplo usando el método de Gauss con pivotaje total con condición de parada

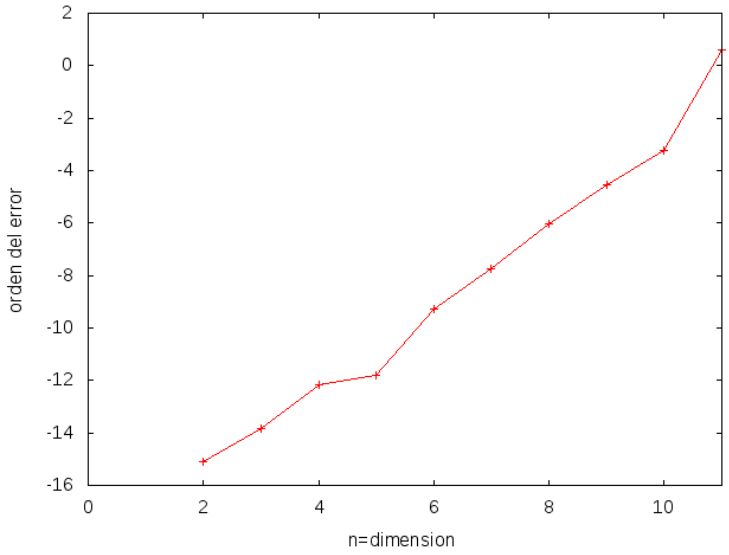


Figura 5: Error del primer ejemplo en escala logarítmica usando el método de Gauss con pivotaje total

De nuevo, es hasta dimensión 11 donde el método funciona y a partir de allí hay algún pivote problemático. Como antes, parece que el método funciona bien aunque cada vez el error va perdiendo precisión, considerándose buenos datos exceptuando para dimensión 10 donde se puede considerar aceptable. En dimensión 11 nos da un mal resultado que podría deberse a que algún pivote es muy pequeño pero no lo suficiente para parar el proceso. Arreglando el programa para que no pare en los pivotes problemáticos, se obtiene la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
12	2,175232322283934	70	$9{,}750142793290344 \cdot 10$
20	$1{,}436165169204416 \cdot 10$	80	$2{,}226706680872628 \cdot 10^{+02}$
30	$1{,}101156117196826 \cdot 10$	90	$8{,}934267505924940 \cdot 10$
40	$9{,}730520127324814 \cdot 10$	100	$5{,}963690294826839 \cdot 10$
50	$3{,}690248962008946 \cdot 10$	1000	$1{,}225971332148075 \cdot 10^{+04}$
60	$8{,}102108877704592 \cdot 10$		

Tabla 11: Error del primer ejemplo usando el método de Gauss con pivotaje total sin condición de parada

Como antes, estos grandes errores provienen del hecho que conforme aumenta la dimensión, los pivotes empiezan a ser más pequeños por lo que al dividirlos conlleva un error que se va acumulando y aumentando significativamente conforme aumenta la dimensión. En tal caso, el primer pivote problemático depende de la dimensión del sistema y se recoge en la tabla siguiente:

n	Primer pivote problemático
12	$a_{10,10}^{(10)} = 3{,}647439602025152 \cdot 10^{-14}$
20	$a_{11,11}^{(11)} = 4{,}546473019709685 \cdot 10^{-13}$
30	$a_{13,13}^{(13)} = 7{,}429422729174636 \cdot 10^{-13}$
40	$a_{14,14}^{(14)} = -7{,}722526219638330 \cdot 10^{-13}$
50	$a_{15,15}^{(15)} = 8{,}664733751218329 \cdot 10^{-13}$
60	$a_{14,14}^{(14)} = -8{,}786036352295864 \cdot 10^{-13}$
70	$a_{15,15}^{(15)} = -8{,}577252995104835 \cdot 10^{-13}$
80	$a_{16,16}^{(16)} = 9{,}952350497795189 \cdot 10^{-13}$
90	$a_{16,16}^{(16)} = -8{,}891822269039415 \cdot 10^{-13}$
100	$a_{16,16}^{(16)} = 9{,}690725626163313 \cdot 10^{-13}$
1000	$a_{22,22}^{(22)} = 9{,}923042276679737 \cdot 10^{-13}$

Tabla 12: Primer pivote que da problemas usando el método de Gauss con pivotaje total

En este caso, para dimensión 1000 obtenemos 978 pivotes problemáticos frente a los 35 del método anterior o los 538 del primero. Con esto, se conlcuye que el pivotaje (parcial) ha permitido disminuir la cantidad de pivotes problemáticos pero que el total lo ha hecho aumentar. Otra observación es que este método, a parte de que no nos ha servido para prolongar la aparición de dicho pivote, ha hecho que para dimensión 11 fallase, cuando antes no era hasta dimensión 12 cuando empezaba a fallar y eso que en dimensión 11 no tenemos ningún problema con los pivotes.

Comparándolo con el método anterior, vemos que hasta dimensión 10 el método funciona con precisión similar. No es comparable para dimensión superior pues ambos métodos nos proporciona un dato erróneo al tener como pivote un número muy pequeño. Aunque si comparamos la tabla sin la condición de parada, se puede observar como para n grande, dicho método mejora a los anteriores.

Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307201749433826 \cdot 10^{-04}$
4	$2{,}755735094107303 \cdot 10^{-03}$	900	$2{,}920285088953616 \cdot 10^{-04}$
9	$2{,}158044791560097 \cdot 10^{-03}$	1600	$2{,}208664522342280 \cdot 10^{-04}$
16	$1{,}758522566483848 \cdot 10^{-03}$	2500	$1{,}775698634802037 \cdot 10^{-04}$
25	$1{,}479488029212956 \cdot 10^{-03}$	3600	$1{,}484428661763759 \cdot 10^{-04}$
36	$1{,}275238805383861 \cdot 10^{-03}$	4900	$1{,}274942602006119 \cdot 10^{-04}$
49	$1{,}119809922150866 \cdot 10^{-03}$	6400	$1{,}116929646818238 \cdot 10^{-04}$
64	$9{,}977841775959493 \cdot 10^{-04}$	8100	$9{,}930645666773320 \cdot 10^{-05}$
81	$8{,}995373727250511 \cdot 10^{-04}$	10000	$8{,}935336501480265 \cdot 10^{-05}$
100	$8{,}187854974974448 \cdot 10^{-04}$		

Tabla 13: Error del segundo ejemplo usando el método de Gauss con pivotaje total

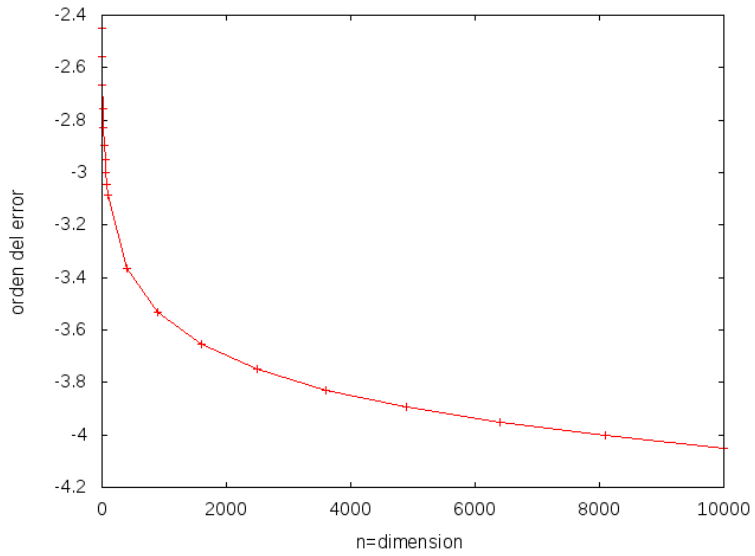


Figura 6: Error del segundo ejemplo en escala logarítmica usando el método de Gauss con pivotaje total

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Según el contexto, el orden de 10^{-4} o 10^{-3} puede ser un error demasiado grande númericamente. Con esto podemos concluir que el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar, cosa extraña porque este método tendría que mejorarlos pues tal y como hemos dicho, este método es una mejora del anterior (y también del primero).
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplo de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}260912328763753 \cdot 10^{-11}$	1,000000000000586
2	100	$3{,}432323085850499 \cdot 10^{-13}$	8,889579542351672
3	100	$3{,}249871443899811 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$6{,}460111323633703 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$2{,}946809511707026 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$4{,}440508779251193 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$3{,}321151869338771 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$3{,}773065118678162 \cdot 10^{-13}$	7,673291683578189
9	100	$3{,}024359529318828 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$4{,}185912351508920 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 14: Error de 10 sistema aleatorios usando el método de Gauss con pivotaje total

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los métodos anteriores, se puede observar nos proporcionan datos similares.

- Descomposición LU:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	7	$3{,}538230583168581 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$3{,}429476607677535 \cdot 10^{-07}$
3	$2{,}583386637281807 \cdot 10^{-14}$	9	$2{,}878010266216586 \cdot 10^{-05}$
4	$3{,}546651024979795 \cdot 10^{-13}$	10	$7{,}098628617533018 \cdot 10^{-04}$
5	$2{,}561350072970022 \cdot 10^{-11}$	11	$6{,}259629404497661 \cdot 10^{-03}$
6	$1{,}416113100575318 \cdot 10^{-09}$	12	$4{,}986368981592573 \cdot 10^{-01}$

Tabla 15: Error del primer ejemplo usando la descomposición LU con condición de parada

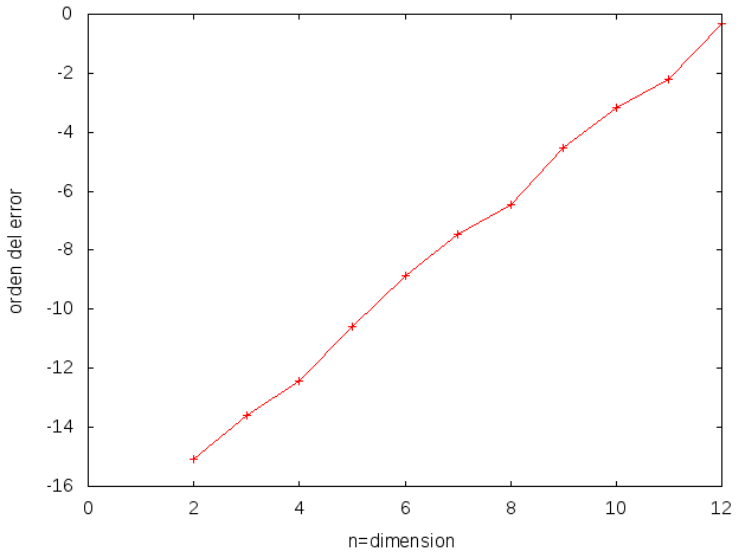


Figura 7: Error del primer ejemplo en escala logarítmica usando la descomposición LU

El resultado obtenido no debe sorprendernos pues la factorización LU proviene de aplicar el método de Gauss a nuestra matriz y guardar los multiplicadores. Es por ello que ambos métodos coincidirán en los pivotes. Por tanto, ambos tendrán exactamente los mismos pivotes problemáticos. Así pues si gauss funciona hasta dimensión 12, LU también ha de funcionar hasta dicha dimensión. Arreglando el programa para que no pare en los pivotes problemáticos, obtenemos la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
13	5,850858557997615	70	$7{,}447539112950283 \cdot 10^{+02}$
20	$4{,}404928349210429 \cdot 10^{+03}$	80	$6{,}202478103040467 \cdot 10^{+03}$
30	$2{,}148162221167840 \cdot 10^{+02}$	90	$2{,}124215890734548 \cdot 10^{+03}$
40	$3{,}073733605070246 \cdot 10^{+02}$	100	$4{,}442668501406128 \cdot 10^{+04}$
50	$5{,}665114573404675 \cdot 10^{+02}$	1000	$1{,}272829153069753 \cdot 10^{+06}$
60	$1{,}561414277553170 \cdot 10^{+03}$		

Tabla 16: Error del primer ejemplo usando la descomposición LU sin condición de parada

Así, dicho método no es necesario compararlo con Gauss puesto que funcionan de la misma manera y por tanto tienen precisión similar. Comparaádolo con los otros, vemos que hasta dimensión 10 el método funciona con precisión similar, para dimensión 11 el método funciona con precisión similar al método de gauss con pivotaje pero mejora al de gauss con pivotaje total y finalmente para dimensión 12 el método funciona mejor.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202391026803 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293663919965 \cdot 10^{-04}$
9	$2{,}158044791560270 \cdot 10^{-03}$	1600	$2{,}208714582532005 \cdot 10^{-04}$
16	$1{,}758522566483077 \cdot 10^{-03}$	2500	$1{,}775893530821708 \cdot 10^{-04}$
25	$1{,}479488029213190 \cdot 10^{-03}$	3600	$1{,}484882119793707 \cdot 10^{-04}$
36	$1{,}275238805384172 \cdot 10^{-03}$	4900	$1{,}275804263696784 \cdot 10^{-04}$
49	$1{,}119809922149617 \cdot 10^{-03}$	6400	$1{,}118330094699110 \cdot 10^{-04}$
64	$9{,}977841776039741 \cdot 10^{-04}$	8100	$9{,}954537043274174 \cdot 10^{-05}$
81	$8{,}995373727266863 \cdot 10^{-04}$	10000	$8{,}969010124347698 \cdot 10^{-05}$
100	$8{,}187854975045424 \cdot 10^{-04}$		

Tabla 17: Error del segundo ejemplo usando la descomposición LU

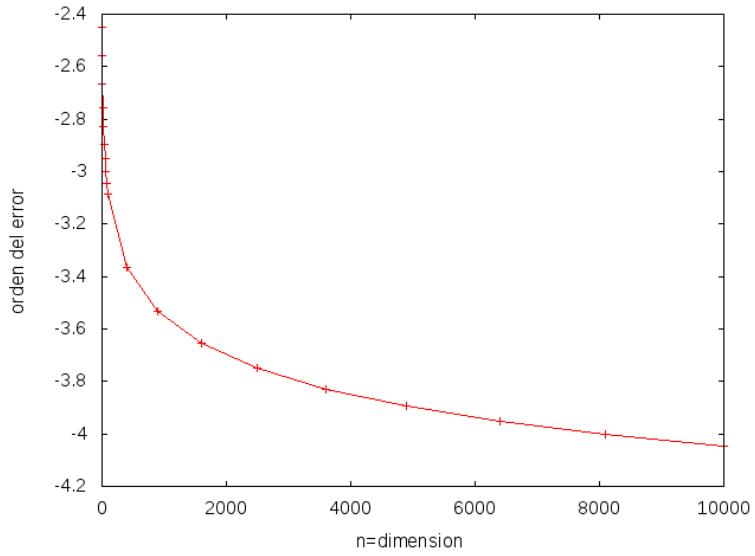


Figura 8: Error del segundo ejemplo en escala logarítmica usando la descomposición LU

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos suponer que el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}133776765103534 \cdot 10^{-11}$	$1{,}000000000000586$
2	100	$2{,}625666827957787 \cdot 10^{-13}$	$8{,}889579542351672$
3	100	$3{,}031655943754083 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$5{,}294602102962564 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$2{,}404735225424495 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$3{,}852880223440721 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$2{,}996640767979772 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$2{,}821512919017008 \cdot 10^{-13}$	$7{,}673291683578189$
9	100	$3{,}269016678667130 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$3{,}514928148739059 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 18: Error de 10 sistema aleatorios usando la descomposición LU

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como nos proporcionan datos similares.

- Descomposición LU con pivotaje:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	$0{,}000000000000000$	7	$1{,}005715972062577 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$1{,}390168685214524 \cdot 10^{-06}$
3	$1{,}216188388897624 \cdot 10^{-15}$	9	$1{,}131578546114154 \cdot 10^{-05}$
4	$2{,}807299850305661 \cdot 10^{-13}$	10	$1{,}909802455870742 \cdot 10^{-03}$
5	$5{,}406613324482790 \cdot 10^{-13}$	11	$2{,}014429457943915 \cdot 10^{-02}$
6	$1{,}093950943578246 \cdot 10^{-09}$		

Tabla 19: Error del primer ejemplo usando la descomposición LU con pivotaje con condición de parada

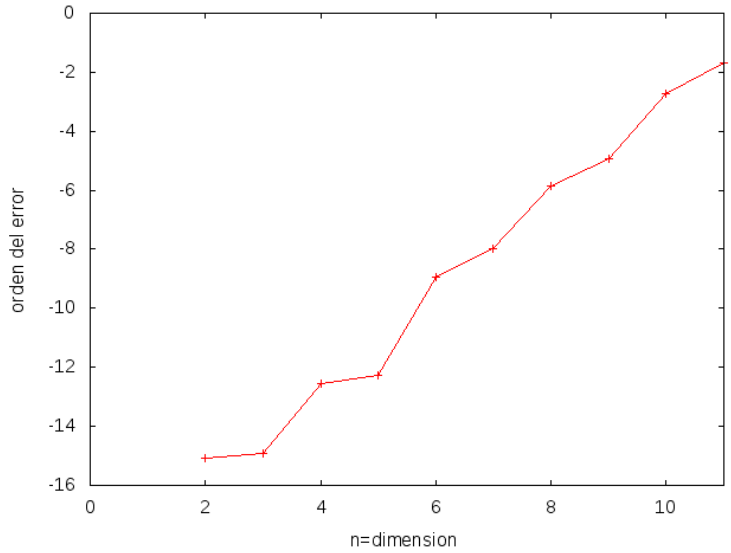


Figura 9: Error del primer ejemplo en escala logarítmica usando la descomposición LU con pivotaje

Como antes, el resultado obtenido no nos ha de sorprender pues la factorización LU con pivotaje proviene de aplicar el método de Gauss con pivotaje a nuestra matriz y guardar los multiplicadores. Es por ello que ambos métodos tendrán los mismos errores en los pivotes y por tanto los pivotes problemáticos coincidirán. Así, si gauss con pivotaje nos funcionaba hasta dimensión 11, LU con pivotaje también nos ha de funcionar hasta dicha dimensión. Arreglando el programa para que no pare en los pivotes problemáticos, obtenemos la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
12	5,244122859131280	70	$5{,}257540460794823 \cdot 10^{+03}$
20	$1{,}036072336430250 \cdot 10^{+03}$	80	$6{,}936984366744055 \cdot 10^{+03}$
30	$1{,}218850532437493 \cdot 10^{+04}$	90	$1{,}182560765346073 \cdot 10^{+04}$
40	$2{,}791743338206478 \cdot 10^{+03}$	100	$1{,}480118168753246 \cdot 10^{+03}$
50	$6{,}025376450397243 \cdot 10^{+03}$	1000	$1{,}323731434148377 \cdot 10^{+08}$
60	$5{,}707452089567866 \cdot 10^{+04}$		

Tabla 20: Error del primer ejemplo usando la descomposición LU con pivotaje sin condición de parada

Así, dicho método no cal compararlo con gauss con pivotaje puesto que funcionan de la misma manera y por tanto tienen precisión similar. Tampoco cal compararlo con gauss y con LU pues la comparación de este método con el de gauss y con LU es equivalente a la comparación entre gauss y gauss con pivotaje que ya lo hemos hecho. Comparandolo con el método restante, vemos que hasta dimensión 10 el método funciona con precisión similar, para dimensión 11 el método funciona mejor.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202391026803 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293663919965 \cdot 10^{-04}$
9	$2{,}158044791560270 \cdot 10^{-03}$	1600	$2{,}208714582532005 \cdot 10^{-04}$
16	$1{,}758522566483077 \cdot 10^{-03}$	2500	$1{,}775893530821708 \cdot 10^{-04}$
25	$1{,}479488029213190 \cdot 10^{-03}$	3600	$1{,}484882119793707 \cdot 10^{-04}$
36	$1{,}275238805384172 \cdot 10^{-03}$	4900	$1{,}275804263696784 \cdot 10^{-04}$
49	$1{,}119809922149617 \cdot 10^{-03}$	6400	$1{,}118330094699110 \cdot 10^{-04}$
64	$9{,}977841776039741 \cdot 10^{-04}$	8100	$9{,}954537043274174 \cdot 10^{-05}$
81	$8{,}995373727266863 \cdot 10^{-04}$	10000	$8{,}969010124347698 \cdot 10^{-05}$
100	$8{,}187854975045424 \cdot 10^{-04}$		

Tabla 21: Error del segundo ejemplo usando la descomposición LU con pivotaje

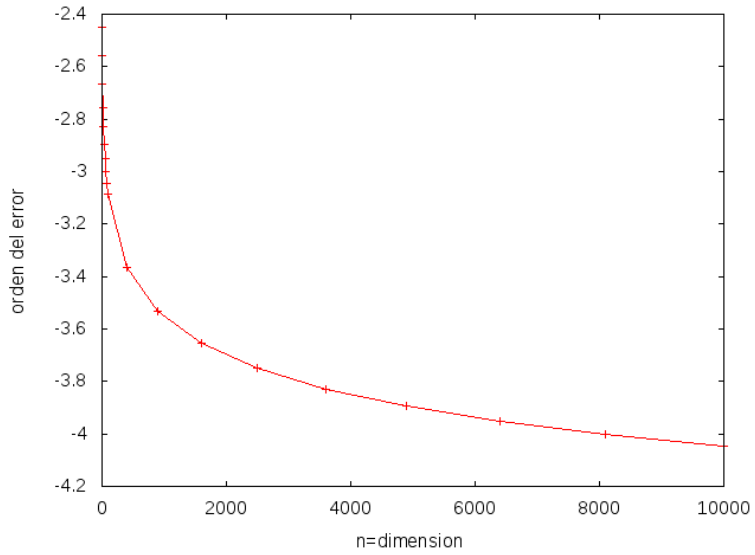


Figura 10: Error del segundo ejemplo en escala logarítmica usando la descomposición LU con pivotaje

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos concluir que para dicho ejemplo, el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anterior, vemos que el método funciona con precisión similar, cosa extraña porque este método tendría que mejorar al anterior pues tal y como hemos dicho, este método es una mejora.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}133776765103534 \cdot 10^{-11}$	1,000000000000586
2	100	$2{,}625666827957787 \cdot 10^{-13}$	8,889579542351672
3	100	$3{,}031655943754083 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$5{,}294602102962564 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$2{,}404735225424495 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$3{,}852880223440721 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$2{,}996640767979772 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$2{,}821512919017008 \cdot 10^{-13}$	7,673291683578189
9	100	$3{,}269016678667130 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$3{,}514928148739059 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 22: Error de 10 sistema aleatorios usando la descomposición LU con pivotaje

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como nos proporcionan datos similares.

- Crout:

En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	7	$4{,}933449634191901 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$1{,}586308949738169 \cdot 10^{-07}$
3	$4{,}965068306494546 \cdot 10^{-16}$	9	$7{,}656322409445531 \cdot 10^{-05}$
4	$9{,}086140387614875 \cdot 10^{-13}$	10	$6{,}108985292405508 \cdot 10^{-04}$
5	$9{,}904321860007972 \cdot 10^{-12}$	11	$1{,}271074001963077 \cdot 10^{-02}$
6	$4{,}681546662013314 \cdot 10^{-10}$		

Tabla 23: Error del primer ejemplo usando crout con condición de parada

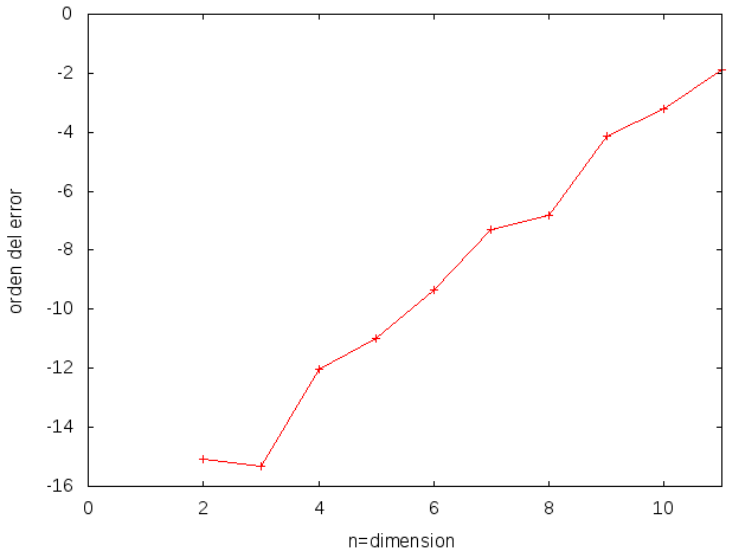


Figura 11: Error del primer ejemplo en escala logarítmica usando Crout

Tal y como se puede observar, el método funciona hasta dimensión 12, donde a partir de allí el proceso finaliza por tener algún pivote problemático aunque en tal caso, no lo necesitamos para calcular el multiplicador correspondiente sino porque en el algoritmo sale diviendo. De entrada, parece que el método funciona bien aunque cada vez el error va perdiendo precisión, considerándose buenos datos exceptuando para dimensión 10, 11 y 12 donde se pueden considerar aceptables, pues según hemos dicho el orden de 10^{-4} o 10^{-2} puede ser un error demasiado grande númericamente. Arreglando el programa para que no pare en los pivotes problemáticos, se puede observar la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
12	$5{,}817043338782428 \cdot 10^{-01}$	20	$8{,}083198505835021 \cdot 10$
30	$3{,}766553648465472 \cdot 10^{+03}$		

Tabla 24: Error del primer ejemplo usando Crout sin condición de parada

La dimensión 12, en tal caso, nos proporciona un resultado aceptable a pesar que el proceso finaliza por obtener un pivote problemático aunque no lo suficientemente pequeño para aumentar el error cometido al dividir. El resto ya obtenemos grandes errores que provienen del hecho que los pivotes empiezan a ser más pequeños por lo que al dividirlos conlleva a un error que se va acumulando y aumentando significativamente conforma aumenta la dimensión. El motivo por el cual hace que dicho método no funcione para dimensión más grande que 40 se resume si miramos todos los pivotes problemáticos resumidos en la siguiente tabla:

Primer pivote problemático	
$a_{11,11} = 8{,}956030361773060 \cdot 10^{-14}$	$a_{26,26} = 1{,}918604164430349 \cdot 10^{-15}$
$a_{12,12} = 2{,}588207426157396 \cdot 10^{-15}$	$a_{27,27} = -3{,}642919299551295 \cdot 10^{-16}$
$a_{13,13} = -6{,}744604874597826 \cdot 10^{-14}$	$a_{28,28} = 3{,}469446951953614 \cdot 10^{-18}$
$a_{14,14} = 6{,}106226635438361 \cdot 10^{-15}$	$a_{29,29} = 7{,}563394355258879 \cdot 10^{-16}$
$a_{15,15} = 9{,}645062526431047 \cdot 10^{-16}$	$a_{30,30} = -3{,}001071613439876 \cdot 10^{-15}$
$a_{16,16} = 3{,}289035710452026 \cdot 10^{-15}$	$a_{31,31} = 9{,}922618282587337 \cdot 10^{-16}$
$a_{17,17} = -2{,}040034807748725 \cdot 10^{-15}$	$a_{32,32} = 8{,}326672684688674 \cdot 10^{-17}$
$a_{18,18} = 1{,}467576060676379 \cdot 10^{-15}$	$a_{33,33} = -7{,}806255641895632 \cdot 10^{-17}$
$a_{19,19} = 1{,}013078509970455 \cdot 10^{-15}$	$a_{34,34} = -1{,}561251128379126 \cdot 10^{-17}$
$a_{20,20} = 9{,}159339953157541 \cdot 10^{-16}$	$a_{35,35} = -1{,}701763729933248 \cdot 10^{-15}$
$a_{21,21} = 1{,}283695372222837 \cdot 10^{-16}$	$a_{36,36} = 1{,}899522206194604 \cdot 10^{-15}$
$a_{22,22} = -8{,}916478666520788 \cdot 10^{-16}$	$a_{37,37} = 1{,}092875789865388 \cdot 10^{-16}$
$a_{23,23} = -4{,}510281037539698 \cdot 10^{-17}$	$a_{38,38} = 3{,}816391647148976 \cdot 10^{-17}$
$a_{24,24} = 1{,}207367539279858 \cdot 10^{-15}$	$a_{39,39} = 0{,}000000000000000$
$a_{25,25} = -1{,}908195823574488 \cdot 10^{-16}$	

Tabla 25: Pivote que da problemas usando el método de crout

Dichos pivotes problemáticos son los mismos para cualquier dimensión siempre que este definido (por ejemplo para dimensión 20, obtendriamos de la tabla los elementos hasta $a_{19,19}$) o para dimensión superior a 40 obtendriamos la misma tabla). Como $a_{39,39} = 0{,}000000000000000$, cuando lo realiza el programa, el cociente con denominador $a_{39,39}$ da infinito por ser cero. Es por ello que el error se acumula y termina siendo infinito.
Comparándolo con los métodos anteriores, vemos que hasta dimensión 10 tiene la misma precisión, para dimensión 11 tiene la misma precisión que gauss, gauss con pivotaje, LU y LU con pivotaje pero mejora al de gauss con pivotaje total y finalmente para dimensión superior no es comparable porque este método tiene pivotes problematicos que hacen que el cociente del método tenga errores que se prolongan sobre la solución.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202390554243 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293729380054 \cdot 10^{-04}$
9	$2{,}158044791559805 \cdot 10^{-03}$	1600	$2{,}208714853478128 \cdot 10^{-04}$
16	$1{,}758522566483525 \cdot 10^{-03}$	2500	$1{,}775894303562101 \cdot 10^{-04}$
25	$1{,}479488029213020 \cdot 10^{-03}$	3600	$1{,}484883939716773 \cdot 10^{-04}$
36	$1{,}275238805384907 \cdot 10^{-03}$	4900	$1{,}275807954795979 \cdot 10^{-04}$
49	$1{,}119809922150104 \cdot 10^{-03}$	6400	$1{,}118336625460466 \cdot 10^{-04}$
64	$9{,}977841775992815 \cdot 10^{-04}$	8100	$9{,}954644539848876 \cdot 10^{-05}$
81	$8{,}995373727222194 \cdot 10^{-04}$	10000	$8{,}969178208236188 \cdot 10^{-05}$
100	$8{,}187854975011871 \cdot 10^{-04}$		

Tabla 26: Error del segundo ejemplo usando Crout

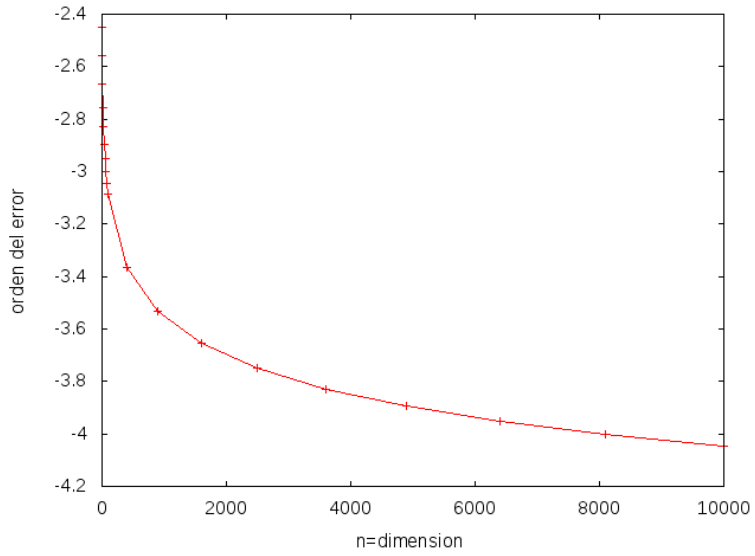


Figura 12: Error del segundo ejemplo en escala logarítmica usando Crout

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos observar que el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$1{,}969919143129036 \operatorname{cot} 10^{-13}$	1,000000000000586
2	100	$2{,}179881322156490 \operatorname{cot} 10^{-13}$	8,889579542351672
3	100	$2{,}072629730387582 \operatorname{cot} 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$3{,}331530340018005 \operatorname{cot} 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$1{,}620877782028720 \operatorname{cot} 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$2{,}335809251424072 \operatorname{cot} 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$2{,}514777086237223 \operatorname{cot} 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$2{,}046982504940873 \operatorname{cot} 10^{-13}$	7,673291683578189
9	100	$1{,}978761628861902 \operatorname{cot} 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$2{,}228335857357574 \operatorname{cot} 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 27: Error de 10 sistema aleatorios usando el método de Crout

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como, excepto el primer sistema que nos mejora, ambos nos proporcionan datos similares.

- Doolittle:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	7	$4{,}913931146305045 \cdot 10^{-08}$
2	$8{,}005932084973442 \cdot 10^{-16}$	8	$8{,}189166420875987 \cdot 10^{-07}$
3	$2{,}812175416797580 \cdot 10^{-14}$	9	$9{,}838435099741797 \cdot 10^{-06}$
4	$3{,}125508724290682 \cdot 10^{-13}$	10	$1{,}081288640267092 \cdot 10^{-04}$
5	$2{,}362692325075402 \cdot 10^{-11}$	11	$3{,}051641850074314 \cdot 10^{-02}$
6	$2{,}158289969099849 \cdot 10^{-09}$		

Tabla 28: Error del primer ejemplo usando doolittle con condición de parada

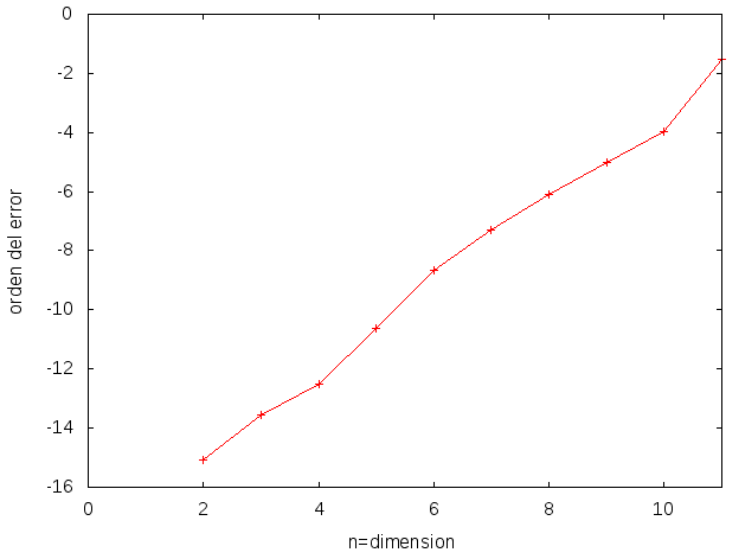


Figura 13: Error del primer ejemplo en escala logarítmica usando Doolittle

Tal y como pasaba en Crout, el método funciona hasta dimensión 12, donde a partir de allí el proceso finaliza por tener algún pivote problemático que como hemos dicho, en tal caso, lo necesitamos porque en el algoritmo sale diviendo. De entrada, parece que el método funciona bien aunque cada vez el error va perdiendo precisión, considerandose buenos datos exceptuando para dimensión 10, 11 y 12 donde se pueden considerar aceptablea, pues según hemos dicho el orden de 10^{-4} o 10^{-2} puede ser un error demasiado grande númericamente. Arreglando el programa para que no pare en los pivotes problemáticos, se puede observar la siguiente tabla:

Ejemplo 1			
n	Error	n	Error
12	$9{,}864969586626992 \cdot 10^{-01}$	20	$6{,}306572550963190 \cdot 10$
30	$5{,}110546125245776 \cdot 10^{+03}$		

Tabla 29: Error del primer ejemplo usando Doolittle

De nuevo, la dimensión 12, nos proporciona un resultado aceptable a pesar que el proceso finalize, lo que significa que obtenemos un pivote problemático pero no lo suficientemente pequeño para aumentar el error cometido al dividir. El resto ya obtenemos grandes errores que provienen del hecho que los pivotes empiezan a ser más pequeños por lo que al dividirlos conlleva un error que se va acumulando y aumentando significativamente conforma aumenta la dimensión. Algo muy curioso, es que los pivotes problemáticos son los mismos que en Crout, por lo que explica porque dicho método no funciona para dimensión superior a 40 (es decir, como antes $a_{39,39}$ es cero, por lo que cuando el programa realiza el cociente con denominador $a_{39,39}$ da infinito por lo que se acumula y obtenemos infinito como solución).
Comparándolo con los métodos anteriores, vemos que tiene una precisión similar a Crout, lo que hace que la comparativa entre Doolittle y cualquier método sea la misma que Crout y dicho método.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	400	$4{,}307202390539914 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	900	$2{,}920293729378026 \cdot 10^{-04}$
9	$2{,}158044791560131 \cdot 10^{-03}$	1600	$2{,}208714853450231 \cdot 10^{-04}$
16	$1{,}758522566483177 \cdot 10^{-03}$	2500	$1{,}775894303526614 \cdot 10^{-04}$
25	$1{,}479488029212829 \cdot 10^{-03}$	3600	$1{,}484883939688917 \cdot 10^{-04}$
36	$1{,}275238805384883 \cdot 10^{-03}$	4900	$1{,}275807954823359 \cdot 10^{-04}$
49	$1{,}119809922151440 \cdot 10^{-03}$	6400	$1{,}118336625265205 \cdot 10^{-04}$
64	$9{,}977841775995629 \cdot 10^{-04}$	8100	$9{,}954644542277520 \cdot 10^{-05}$
81	$8{,}995373727234049 \cdot 10^{-04}$	10000	$8{,}969178207250926 \cdot 10^{-05}$
100	$8{,}187854975027096 \cdot 10^{-04}$		

Tabla 30: Error del segundo ejemplo usando Doolittle

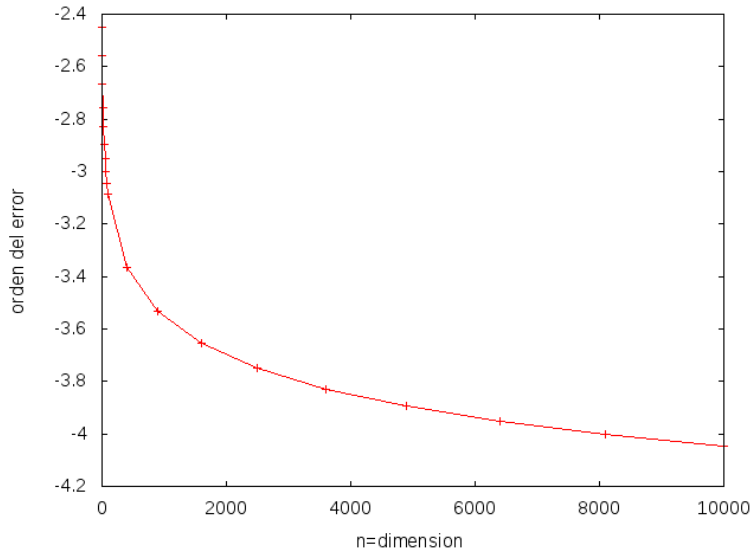


Figura 14: Error del segundo ejemplo en escala logarítmica usando Doolittle

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos concluir que para dicho ejemplo, el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$1{,}965656208217586 \cdot 10^{-13}$	1,000000000000586
2	100	$2{,}150472555087284 \cdot 10^{-13}$	8,889579542351672
3	100	$2{,}066359548182499 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$3{,}123107811522473 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$1{,}830447750906109 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$2{,}463047824984093 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$2{,}504806470562982 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$2{,}112654674428332 \cdot 10^{-13}$	7,673291683578189
9	100	$1{,}997324580191612 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$2{,}152907776660317 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 31: Error de 10 sistema aleatorios usando Doolittle

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como nos proprcionan datos similares.

- Descomposición QR por Householder:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	20	$3{,}227612307799495 \cdot 10^{+02}$
2	$2{,}286089127109703 \cdot 10^{-15}$	30	$4{,}809240408981456 \cdot 10^{+02}$
3	$3{,}181132864214931 \cdot 10^{-14}$	40	$1{,}251071131688401 \cdot 10^{+03}$
4	$6{,}487985008513230 \cdot 10^{-13}$	50	$1{,}282624965495723 \cdot 10^{+03}$
5	$2{,}505229891126522 \cdot 10^{-11}$	60	$2{,}212887760320717 \cdot 10^{+03}$
6	$1{,}258244228606995 \cdot 10^{-09}$	70	$6{,}958004855165019 \cdot 10^{+03}$
7	$1{,}945022064277736 \cdot 10^{-08}$	80	$1{,}070490196131989 \cdot 10^{+04}$
8	$3{,}266705943172173 \cdot 10^{-06}$	90	$1{,}711686506966249 \cdot 10^{+04}$
9	$1{,}562681835622997 \cdot 10^{-05}$	100	$1{,}133741734577246 \cdot 10^{+04}$
10	$1{,}242670936879134 \cdot 10^{-03}$	1000	$2{,}156821239796138 \cdot 10^{+06}$

Tabla 32: Error del primer ejemplo usando la descomposición QR por Householder

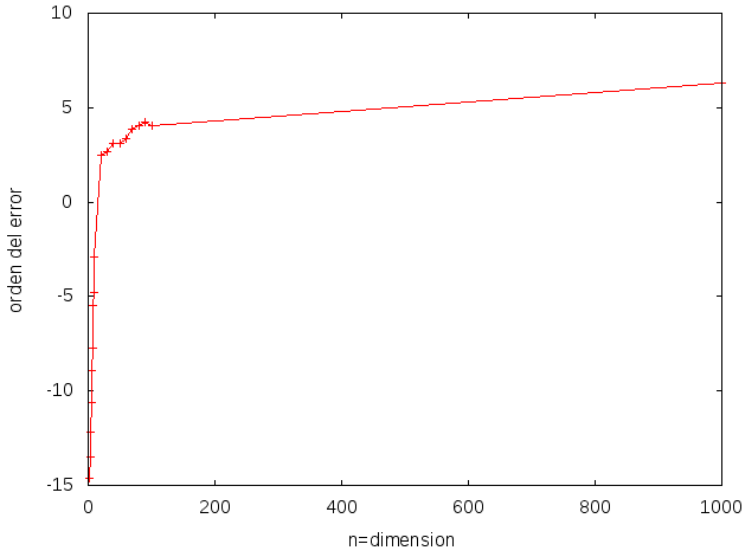


Figura 15: Error del primer ejemplo en escala logarítmica usando la descomposición QR por Householder

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión. Se puede observar que hasta dimensión 8 el método es bueno y que para dimensión 9 y 10 los resultados empiezar a empeorar pero aún los podríamos dar por aceptables. Para dimensión superior, podemos concluir que estos órdenes del error, incluido el 10^{+2}, son ya preocupantes y no aceptables por lo grandes que empiezan a ser. Con esto podemos concluir, que este método empieza a fallar tanto que no podemos aceptar la solución dada por el programa. Resumiendo este método conforme aumenta la dimensión va siendo cada vez menos aconsejable.
Comparándolo con los métodos anteriores, vemos que para dimensión pequeña el método funciona con precisión similar, aunque según que dimensión empeora a la descomposición LU y mejora a Gauss y Gauss con pivotaje, pero conforme aumenta la dimensión, el método mejora a Gauss con pivotaje pero empeora al método de Gauss y a la descomposición LU. A pesar de

esto, es el método que mejor aproxima (aunque alejada) de la solución real en dimensión 1000 de los vistos hasta ahora.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	49	$2{,}798950868304349 \cdot 10^{-03}$
4	$5{,}804379915486936 \cdot 10^{-03}$	64	$2{,}363330384751206 \cdot 10^{-03}$
9	$5{,}891625410437240 \cdot 10^{-03}$	81	$2{,}022315160496629 \cdot 10^{-03}$
16	$4{,}979242565975857 \cdot 10^{-03}$	100	$1{,}751879090420742 \cdot 10^{-03}$
25	$4{,}083943108044876 \cdot 10^{-03}$	400	$6{,}556687833277926 \cdot 10^{-04}$
36	$3{,}360751265883315 \cdot 10^{-03}$	900	$3{,}769950474119915 \cdot 10^{-04}$

Tabla 33: Error del segundo ejemplo usando la descomposición QR por Householder

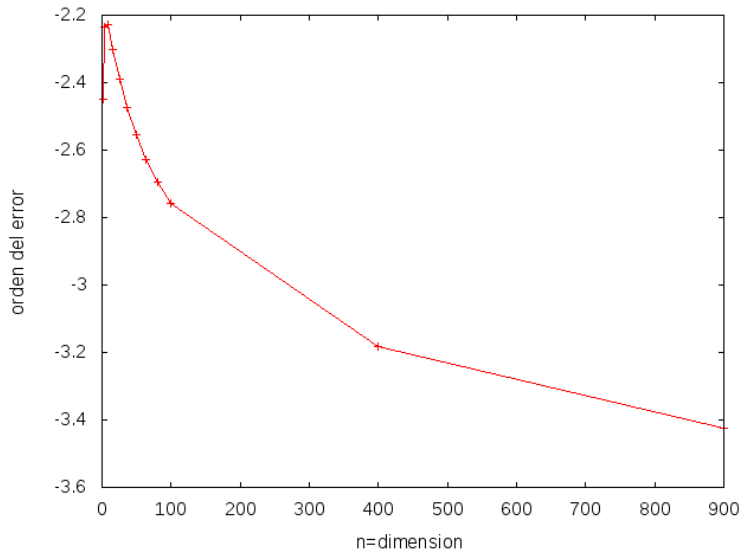


Figura 16: Error del segundo ejemplo en escala logarítmica usando la descomposición QR por Householder

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión pero en tal caso, el tiempo de ejecución empieza a crecer considerablemete conforme aumenta la dimensión (véase el 2.5). En tal ejemplo, podemos concluir que el método, para cualquier dimensión, es aceptable pero no aconsejable pues obtenemos datos similares en un tiempo excesivo. Dicho método va siendo cada vez más aconsejable conforme aumenta la dimensión por los resultados pero no cuanto al tiempo de ejecución que requiere.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$5{,}486169601981407 \cdot 10^{-03}$	1,000000000000586
2	100	$2{,}204470319622476 \cdot 10^{-12}$	8,889579542351672
3	100	$1{,}689562437695646 \cdot 10^{-12}$	$1{,}045321894829862 \cdot 10$
4	100	$3{,}125307212175455 \cdot 10^{-12}$	$1{,}373886821611351 \cdot 10$
5	100	$2{,}027998537539766 \cdot 10^{-12}$	$1{,}245960228492898 \cdot 10$
6	100	$1{,}751968118189519 \cdot 10^{-12}$	$1{,}130441074930559 \cdot 10$
7	100	$2{,}306836480154588 \cdot 10^{-12}$	$1{,}018530744653610 \cdot 10$
8	100	$2{,}212910116903329 \cdot 10^{-12}$	7,673291683578189
9	100	$1{,}473089862399972 \cdot 10^{-12}$	$1{,}049783566225507 \cdot 10$
10	100	$2{,}308636913546664 \cdot 10^{-12}$	$1{,}203512121483044 \cdot 10$

Tabla 34: Error de 10 sistema aleatorios usando la descomposición QR por Householder

Se puede observar como, exceptuando el primer sistema que nos da un dato aceptable, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como, excepto para el primer sistema que empeora, dicho método nos proprciona datos similares a los anteriores aunque algo más grande.

- Descomposición QR por Givens:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	20	7,937365404613277
2	$9{,}930136612989092 \cdot 10^{-16}$	30	$1{,}215582295434033 \cdot 10^{+02}$
3	$6{,}413400083909417 \cdot 10^{-15}$	40	$2{,}131881746084658 \cdot 10^{+02}$
4	$5{,}954966089875692 \cdot 10^{-13}$	50	$3{,}700962502839930 \cdot 10$
5	$9{,}871638533405162 \cdot 10^{-13}$	60	$9{,}248456834340105 \cdot 10$
6	$8{,}742963515665986 \cdot 10^{-10}$	70	$1{,}483190501902950 \cdot 10^{+02}$
7	$1{,}681306541368410 \cdot 10^{-08}$	80	$8{,}989917912976720 \cdot 10^{+02}$
8	$2{,}322690262960618 \cdot 10^{-06}$	90	$7{,}619017266058752 \cdot 10^{+02}$
9	$5{,}382876000380413 \cdot 10^{-05}$	100	$1{,}077625302028431 \cdot 10^{+03}$
10	$1{,}346923070834607 \cdot 10^{-05}$	1000	$8{,}626833977974479 \cdot 10^{+07}$

Tabla 35: Error del primer ejemplo usando la descomposición QR por Givens

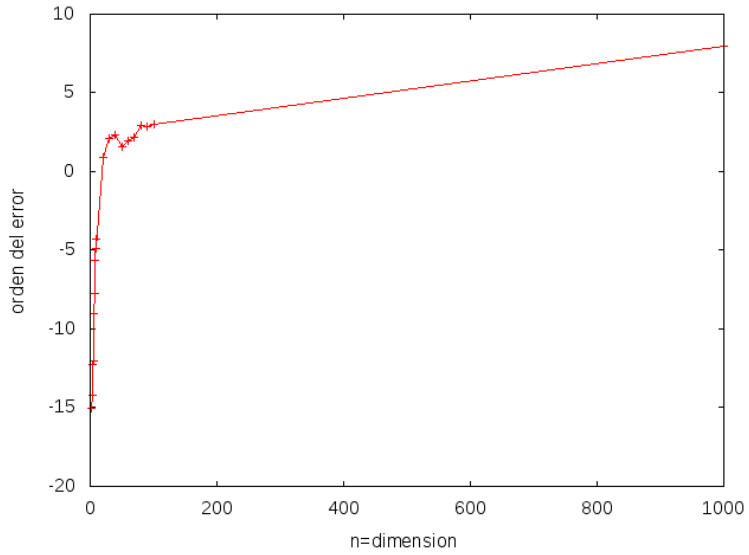


Figura 17: Error del primer ejemplo en escala logarítmica usando la descomposición QR por Givens

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión. Podemos concluir que para dicho ejemplo, hasta dimensión 8 el método es bueno y que para dimensión 9 y 10 los resultados empiezar a empeorar pero aún los podríamos dar por aceptables. Para dimensión superior, podemos concluir que estos órdenes del error, incluido el 10^{+2}, son ya preocupantes y no aceptables por lo grandes que empiezan a ser. Con esto podemos concluir, que este método empieza a fallar tanto que no podemos aceptar la solución dada por el programa. Resumiendo este método conforme aumenta la dimensión va siendo cada vez menos aconsejable.
Comparándolo con los métodos anteriores, vemos que para dimensión pequeña el método funciona con precisión similar pero conforme aumenta la dimensión, el método empeora los anteriores.

Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	n^2	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	81	$8{,}995373727214207 \cdot 10^{-04}$
4	$2{,}755735094107197 \cdot 10^{-03}$	100	$8{,}187854975058886 \cdot 10^{-04}$
9	$2{,}158044791560617 \cdot 10^{-03}$	400	$4{,}307202391503121 \cdot 10^{-04}$
16	$1{,}758522566483904 \cdot 10^{-03}$	900	$2{,}920293729601502 \cdot 10^{-04}$
25	$1{,}479488029214699 \cdot 10^{-03}$	1600	$2{,}208714847941150 \cdot 10^{-04}$
36	$1{,}275238805381339 \cdot 10^{-03}$	2500	$1{,}775894301710724 \cdot 10^{-04}$
49	$1{,}119809922146434 \cdot 10^{-03}$	3600	$1{,}484883929995227 \cdot 10^{-04}$
64	$9{,}977841775926572 \cdot 10^{-04}$		

Tabla 36: Error del segundo ejemplo usando la descomposición QR por Givens

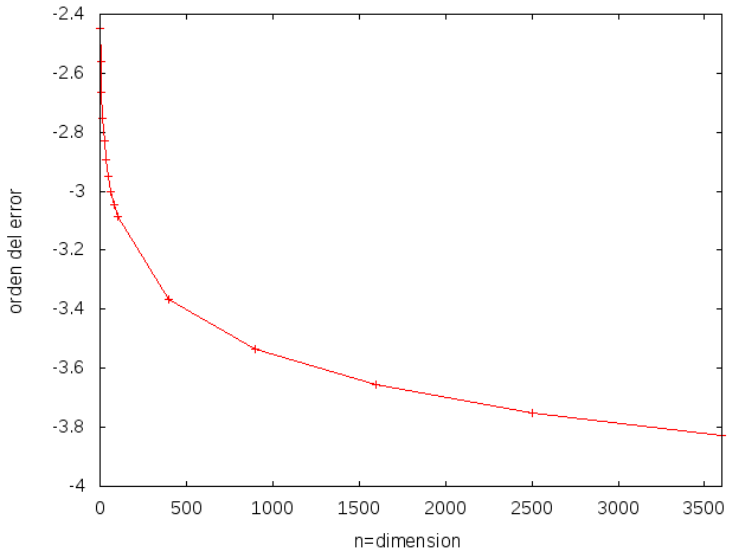


Figura 18: Error del segundo ejemplo en escala logarítmica usando la descomposición QR por Givens

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión pero en tal caso, el tiempo de ejecución empieza a crecer considerablemete conforme aumenta la dimensión (véase el 2.5). Como en el método anterior, podemos oberservar que el método, para cualquier dimensión, es aceptable pero no aconsejable por obtener datos similares en un tiempo mayor, a pesar que aqui el tiempo nos ha permitido hacer alguna dimensión más. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión por los errores obtenidos pero no por el tiempo de ejecución que requiere.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios			
Sistema	n	Error	$\kappa(A)$
1	100	$4{,}121955765619340 \cdot 10^{-13}$	1,000000000000586
2	100	$4{,}619875413302668 \cdot 10^{-13}$	8,889579542351672
3	100	$4{,}895143026555914 \cdot 10^{-13}$	$1{,}045321894829862 \cdot 10$
4	100	$7{,}984710929962769 \cdot 10^{-13}$	$1{,}373886821611351 \cdot 10$
5	100	$4{,}481614663468410 \cdot 10^{-13}$	$1{,}245960228492898 \cdot 10$
6	100	$5{,}901523488512397 \cdot 10^{-13}$	$1{,}130441074930559 \cdot 10$
7	100	$5{,}633297363167522 \cdot 10^{-13}$	$1{,}018530744653610 \cdot 10$
8	100	$4{,}827139546253176 \cdot 10^{-13}$	7,673291683578189
9	100	$4{,}506062765117203 \cdot 10^{-13}$	$1{,}049783566225507 \cdot 10$
10	100	$5{,}185303762016695 \cdot 10^{-13}$	$1{,}203512121483044 \cdot 10$

Tabla 37: Error de 10 sistema aleatorios usando la descomposición QR por Givens

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anteriores, se observa como mejora al anterior pero nos proporciona datos similares al resto, excepto para el primer sistema que nos

proporciona un dato similar a Crout y Doolittle pero que mejora al resto.

Hasta aquí los métodos directos. En tal caso, habían ejemplos donde para cualquier dimensión nos proporcionaba buenos resultados. Sin embargo, en otros, el método perdía eficacia llegando a dar resultados que no tienen que ver con la solución real. Veamos ahora que pasa con los métodos iterativos y de Krylov:

- Gauss-Seidel:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1					
n	Error	Iteraciones	n	Error	Iteraciones
1	0,000000000000000	2	20	$1{,}623670850325427 \cdot 10^{-01}$	1000
2	$2{,}220446049250313 \cdot 10^{-16}$	2	30	$1{,}715286689506031 \cdot 10^{-01}$	1000
3	$5{,}776107378594097 \cdot 10^{-10}$	1000	40	$1{,}765034821252254 \cdot 10^{-01}$	1000
4	$3{,}922918374688335 \cdot 10^{-02}$	1000	50	$1{,}794459074092775 \cdot 10^{-01}$	1000
5	$1{,}051330304826284 \cdot 10^{-01}$	1000	60	$1{,}813512146254031 \cdot 10^{-01}$	1000
6	$1{,}090796260543447 \cdot 10^{-01}$	1000	70	$1{,}827000442766651 \cdot 10^{-01}$	1000
7	$1{,}221167899766702 \cdot 10^{-01}$	1000	80	$1{,}837241250407402 \cdot 10^{-01}$	1000
8	$1{,}296394010055700 \cdot 10^{-01}$	1000	90	$1{,}845396765232202 \cdot 10^{-01}$	1000
9	$1{,}324847766179085 \cdot 10^{-01}$	1000	100	$1{,}852097491192853 \cdot 10^{-01}$	1000
10	$1{,}402209369355631 \cdot 10^{-01}$	1000	1000	$1{,}909763264520113 \cdot 10^{-01}$	1000

Tabla 38: Error del primer ejemplo usando Gauss-Seidel

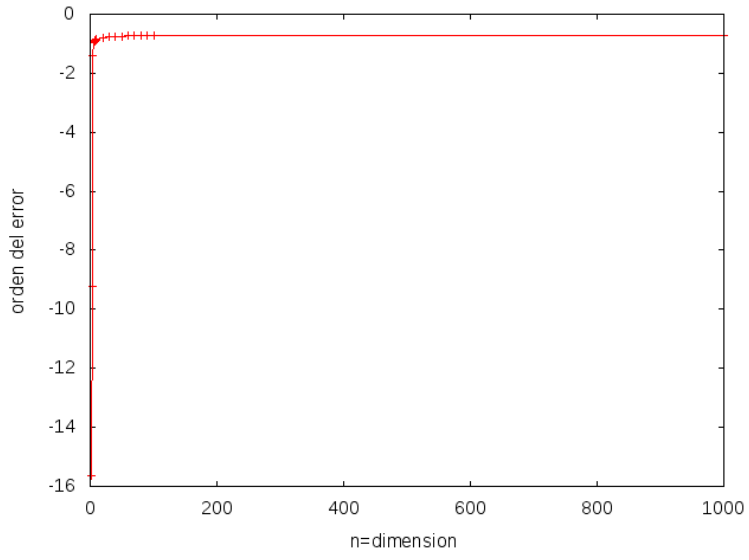


Figura 19: Error del primer ejemplo en escala logarítmica usando Gauss-Seidel

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión hasta llegar a dimensión 4 donde se queda estable. En tal ejemplo, el método es bueno para dimensión 2 pero a partir de dimensión 3 los resultados empiezar a empeorar pero aún

los podriamos dar por aceptables. Comparándolo con los métodos anteriores, vemos que para dimensión pequeña el método empeora (excepto para dimensión 2 donde el método funciona con precisión similar) a los métodos directos, pero que para dimensión superior el método lo mejora, que a pesar de que el orden que nos dé empieze a ser grande, se puede considerar como aceptable.

Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2					
n^2	Error	Iteraciones	n^2	Error	Iteraciones
1	$3{,}568515041634690 \cdot 10^{-03}$	2	400	$4{,}307163317619962 \cdot 10^{-04}$	1000
4	$2{,}755735093861657 \cdot 10^{-03}$	22	900	$8{,}320516548783663 \cdot 10^{-04}$	1000
9	$2{,}158044790986651 \cdot 10^{-03}$	43	1600	$1{,}220011867757886 \cdot 10^{-01}$	1000
16	$1{,}758522564691851 \cdot 10^{-03}$	68	2500	$1{,}217228994861878$	1000
25	$1{,}479488026290024 \cdot 10^{-03}$	99	3600	$4{,}569837510507518$	1000
36	$1{,}275238801639538 \cdot 10^{-03}$	136	4900	$1{,}066287109142656 \cdot 10$	1000
49	$1{,}119809917098712 \cdot 10^{-03}$	178	6400	$1{,}915914464468871 \cdot 10$	1000
64	$9{,}977841707602471 \cdot 10^{-04}$	225	8100	$2{,}943462382602122 \cdot 10$	1000
81	$8{,}995373635931734 \cdot 10^{-04}$	277	10000	$4{,}090479736977755 \cdot 10$	1000
100	$8{,}187854865004193 \cdot 10^{-04}$	335	1000000	$1{,}202461093399270 \cdot 10^{+03}$	1000

Tabla 39: Error del segundo ejemplo usando Gauss-Seidel

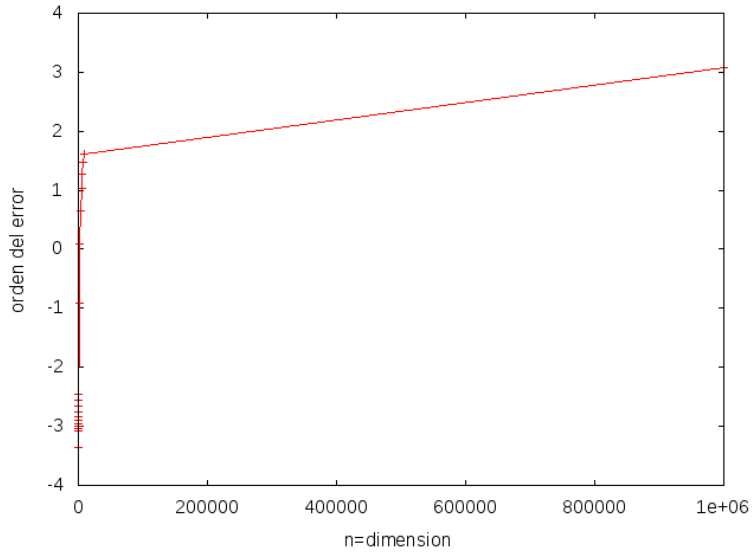


Figura 20: Error del segundo ejemplo en escala logarítmica usando Gauss-Seidel

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión hasta llegar a dimensión 1600 donde empieza a empeorar. Así pues, el método, hasta dimensión 900, es aceptable y para dimensión 1600 sigue siéndolo pero menos. Sin embargo, a partir de dimensión 1600, estos órdenes del error, son ya preocupantes y no aceptables por lo grandes que empiezan a ser. Con esto podemos concluir que este método empieza a fallar tanto que no podemos aceptar la solución dada por

el sistema. Al igual que el primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión, hasta alcanzar la dimensión 900 pero, en contra del primer ejemplo, menos aconsejable conforme aumenta la dimensión a partir de dimensión 1600. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar hasta dimensión 900 pero los empeora a partir de dimensión 1600.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios				
Sistema	n	Error	Iteraciones	$\kappa(A)$
1	100	$1{,}743125113685985 \cdot 10^{-13}$	2	1,000000000000586
2	100	$8{,}312208806111824 \cdot 10^{-13}$	126	8,889579542351672
3	100	$8{,}652830570782423 \cdot 10^{-13}$	144	$1{,}045321894829862 \cdot 10$
4	100	$9{,}288369711565097 \cdot 10^{-13}$	204	$1{,}373886821611351 \cdot 10$
5	100	$9{,}254129697779431 \cdot 10^{-13}$	177	$1{,}245960228492898 \cdot 10$
6	100	$9{,}894028145778797 \cdot 10^{-13}$	160	$1{,}130441074930559 \cdot 10$
7	100	$9{,}912354949284896 \cdot 10^{-13}$	148	$1{,}018530744653610 \cdot 10$
8	100	$7{,}648469865982510 \cdot 10^{-13}$	109	7,673291683578189
9	100	$8{,}491049179755951 \cdot 10^{-13}$	148	$1{,}049783566225507 \cdot 10$
10	100	$9{,}392593443631927 \cdot 10^{-13}$	175	$1{,}203512121483044 \cdot 10$

Tabla 40: Error de 10 sistema aleatorios usando Gauss-Seidel

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anterior, se observa nos proprcionan datos similares a Crout, Doolittle y QR Givens pero que para el resto de métodos los mejora.

- Jacobi:
  Como con 1000 iteraciones máximas, el primer ejemplo obtememos el siguiente resultado:

Ejemplo 1		
n	Error	Iteraciones
1	0,000000000000000	2
2	$7{,}200906537718765 \cdot 10^{-13}$	187

Tabla 41: Error del primer ejemplo usando Jacobi con 1000 iteraciones máximas

lo haremos con 100 iteraciones máximas en el primer ejemplo pero con 1000 iteraciones máximas para el segundo. En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1					
n	Error	Iteraciones	n	Error	Iteraciones
1	0,000000000000000	2	8	$3{,}811549209865807 \cdot 10^{+77}$	100
2	$1{,}132643310253201 \cdot 10^{-07}$	100	9	$2{,}713619226201576 \cdot 10^{+83}$	100
3	$1{,}091526046699491 \cdot 10^{+23}$	100	10	$3{,}729470209156936 \cdot 10^{+88}$	100
4	$3{,}962702756165787 \cdot 10^{+40}$	100	20	$1{,}671426952446189 \cdot 10^{+121}$	100
5	$1{,}441475249349172 \cdot 10^{+53}$	100	30	$4{,}633989392612779 \cdot 10^{+139}$	100
6	$7{,}478765707494496 \cdot 10^{+62}$	100	40	$3{,}724908789274513 \cdot 10^{+152}$	100
7	$7{,}225315885585713 \cdot 10^{+70}$	100			

Tabla 42: Error del primer ejemplo usando Jacobi con 100 iteraciones máximas

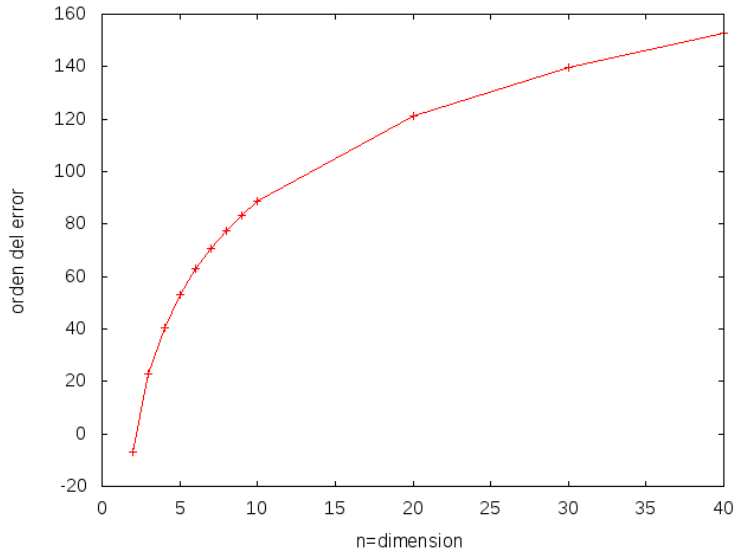


Figura 21: Error del primer ejemplo en escala logarítmica usando Jacobi

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión. Podemos observar que para cualquier dimensión, este método nos da un mal resultado. Comparándolo con los métodos anteriores, vemos que es el método que conforme aumenta la dimensión, la solución dada se aleja cada vez más de la solución real. Es por eso, que podemos decir que es el peor método de los visto hasta ahora.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2					
n^2	Error	Iteraciones	n^2	Error	Iteraciones
1	$3{,}568515041634690 \cdot -03$	2	400	$1{,}389175733089852 \cdot 10^{-04}$	1000
4	$2{,}755735093495641 \cdot 10^{-03}$	42	900	$1{,}926086395112892 \cdot 10^{-01}$	1000
9	$2{,}158044789797700 \cdot 10^{-03}$	82	1600	$2{,}316207148238641$	1000
16	$1{,}758522562933296 \cdot 10^{-03}$	132	2500	$8{,}154881486362017$	1000
25	$1{,}479488023006524 \cdot 10^{-03}$	192	3600	$1{,}728051166177042 \cdot 10$	1000
36	$1{,}275238796525940 \cdot 10^{-03}$	263	4900	$2{,}847914019010896 \cdot 10$	1000
49	$1{,}119809910745977 \cdot 10^{-03}$	345	6400	$4{,}078402512356278 \cdot 10$	1000
64	$9{,}977841628831397 \cdot 10^{-04}$	437	8100	$5{,}361441957791139 \cdot 10$	1000
81	$8{,}995373539552135 \cdot 10^{-04}$	539	10000	$6{,}666630146619772 \cdot 10$	1000
100	$8{,}187854739554412 \cdot 10^{-04}$	651	1000000	$1{,}226710188639807 \cdot 10^{+03}$	1000

Tabla 43: Error del segundo ejemplo usando Jacobi

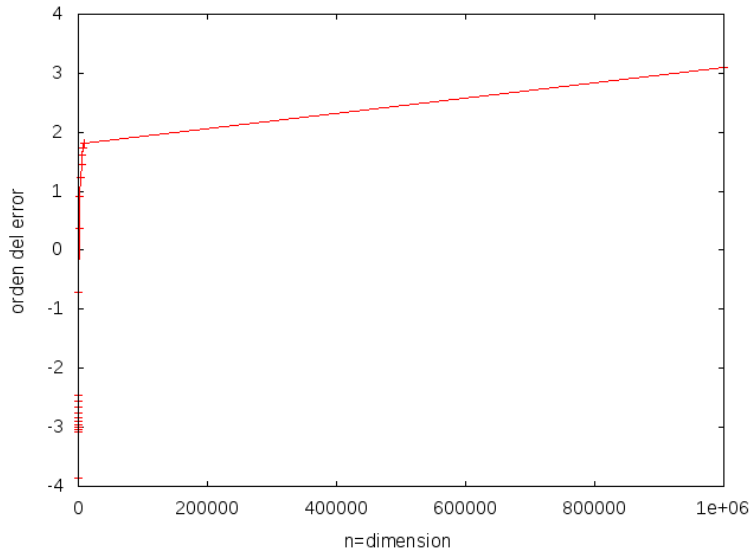


Figura 22: Error del segundo ejemplo en escala logarítmica usando Jacobi

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión hasta llegar a dimensión 1600 donde empieza a empeorar.Podemos concluir que en tal ejemplo, el método, hasta dimensión 900 es aceptable y para dimensión 1600 sigue siéndolo pero menos. Sin embargo, a partir de dimensión 1600, estos órdenes del error, son ya preocupantes y no aceptables por lo grandes que empiezan a ser. Con esto podemos concluir que este método empieza a fallar tanto que no podemos aceptar la solución dada por el sistema. Al igual que el primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión, hasta alcanzar la dimensión 900 pero, en contra del primer ejemplo, menos aconsejable conforme aumenta la dimensión a partir de dimensión 1600. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar hasta dimensión 900 pero los empeora a partir de dimensión 1600 aunque tiene precisión similar a Gauss-Seidel.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios				
Sistema	n	Error	Iteraciones	$\kappa(A)$
1	100	$1{,}743125113685985 \cdot 10^{-13}$	2	1,000000000000586
2	100	$9{,}124291983171751 \cdot 10^{-13}$	242	8,889579542351672
3	100	$9{,}447625527657278 \cdot 10^{-13}$	273	$1{,}045321894829862 \cdot 10$
4	100	$9{,}574274935225161 \cdot 10^{-13}$	400	$1{,}373886821611351 \cdot 10$
5	100	$9{,}480071688579811 \cdot 10^{-13}$	344	$1{,}245960228492898 \cdot 10$
6	100	$9{,}354792943196899 \cdot 10^{-13}$	316	$1{,}130441074930559 \cdot 10$
7	100	$9{,}129861779189446 \cdot 10^{-13}$	290	$1{,}018530744653610 \cdot 10$
8	100	$9{,}529937764037931 \cdot 10^{-13}$	212	7,673291683578189
9	100	$9{,}850863414203370 \cdot 10^{-13}$	284	$1{,}049783566225507 \cdot 10$
10	100	$9{,}214389046461373 \cdot 10^{-13}$	344	$1{,}203512121483044 \cdot 10$

Tabla 44: Error de 10 sistema aleatorios usando Jacobi

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anterior, se observa nos proprcionan datos similares a Crout,

Doolittle, QR Givens y Gauss-Seidel pero que para el resto de métodos los mejora.

- SOR:
  Como bien hemos dicho en la nota anterior, para el segundo ejemplo es fácil saber el w que hay que coger para que nos dé el error más óptimo, variando según la dimensión del sistema, pero el problema viene en el primer ejemplo. En tal caso, así de entrada, no tenemos ningún teorema que nos diga este w óptimo. Es por ello que nos surge la siguiente pregunta "Si no conocemos el w óptimo, ¿Cómo podemos encontrarlo?". La respuesta es que aplicaremos el método con 100 valores distintos de la w, comprendidos entre 1 y 2 (es decir, lo que haremos será empezar con $w = 1$ y le iremos aumentando 0,01 hasta obtener 100 valores distintos comprendidos en el intervalo $[0, 1)$) y entre todos ellos nos quedaremos con el w que tenga el error más pequeño. Así la tabla para el primer ejemplo queda:

Ejemplo 1			
n	Error	w óptima	Iteraciones
1	0,000000000000000	1,000000000000000	1
2	$9{,}171409881229778 \cdot 10^{-14}$	1,470000000000000	37
3	$8{,}496887217546161 \cdot 10^{-13}$	1,690000000000001	195
4	$1{,}674042999869109 \cdot 10^{-07}$	1,790000000000001	1000
5	$1{,}834432197234958 \cdot 10^{-03}$	1,700000000000001	1000
6	$1{,}478555850463555 \cdot 10^{-02}$	1,710000000000001	1000
7	$1{,}443187754363500 \cdot 10^{-02}$	1,720000000000001	1000
8	$1{,}863976010890975 \cdot 10^{-02}$	1,740000000000001	1000
9	$2{,}240387292929920 \cdot 10^{-02}$	1,750000000000001	1000
10	$2{,}070421120278589 \cdot 10^{-02}$	1,770000000000001	1000
20	$5{,}013493399254550 \cdot 10^{-02}$	1,800000000000001	1000
30	$7{,}256050170275473 \cdot 10^{-02}$	1,810000000000001	1000
40	$8{,}244046525687071 \cdot 10^{-02}$	1,810000000000001	1000
50	$8{,}720776149141425 \cdot 10^{-02}$	1,810000000000001	1000
60	$9{,}011087527975531 \cdot 10^{-02}$	1,810000000000001	1000
70	$9{,}223270840460189 \cdot 10^{-02}$	1,810000000000001	1000
80	$9{,}382380926114130 \cdot 10^{-02}$	1,820000000000001	1000
90	$9{,}520523667309923 \cdot 10^{-02}$	1,820000000000001	1000
100	$9{,}645057330109347 \cdot 10^{-02}$	1,820000000000001	1000
1000	$1{,}078437149461118 \cdot 10^{-01}$	1,810000000000001	1000

Tabla 45: Error del primer ejemplo usando SOR con w óptimo

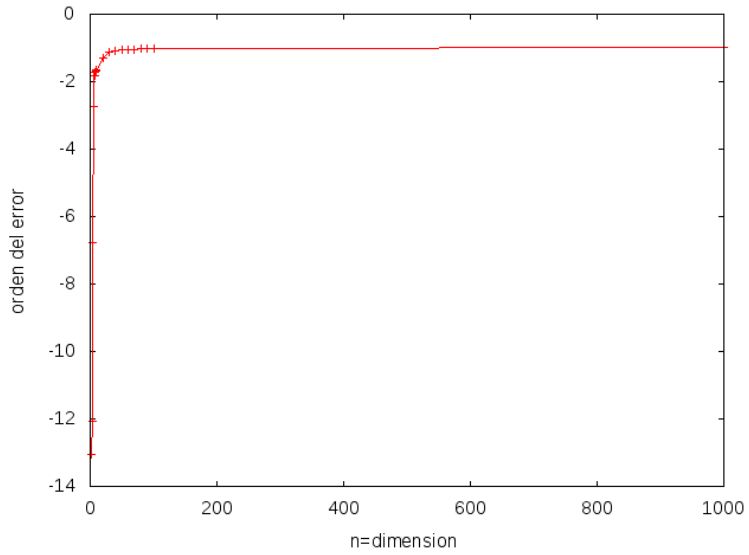


Figura 23: Error del primer ejemplo en escala logarítmica usando SOR

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión. Podemos observar que el método es bueno para dimensión 2 y 3 pero que a partir de dimensión 4 los resultados empiezar a empeorar pero aún los podriamos dar por aceptables. Comparándolo con los métodos anteriores, vemos que para dimensión pequeña el método empeora (excepto para dimensión 2 y 3 donde el método funciona con precisión similar) a los métodos directos pero mejora a Gauss-Seidel y-Jacobi. En cambio, para dimensión superior el método mejora a los vistos hasta ahora (tanto directos como iterativos), que a pesar de que el orden que nos dé empieze a ser grande, se puede considerar como aceptable.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2			
n^2	Error	w óptima	Iteraciones
1	$3{,}568515041634690 \cdot 10^{-03}$	1,000000000000000	1
4	$2{,}755735093711272 \cdot 10^{-03}$	1,630000000000001	64
9	$1{,}988319657683748 \cdot 10^{-03}$	1,990000000000001	1000
16	$1{,}565780409133942 \cdot 10^{-03}$	1,990000000000001	1000
25	$1{,}479488026290024 \cdot 10^{-03}$	1,000000000000000	99
36	$1{,}156037082700683 \cdot 10^{-03}$	1,990000000000001	1000
49	$1{,}017229661288564 \cdot 10^{-03}$	1,990000000000001	1000
64	$9{,}413944574225803 \cdot 10^{-04}$	1,990000000000001	1000
81	$6{,}369314906646601 \cdot 10^{-04}$	1,990000000000001	1000
100	$6{,}797627899198538 \cdot 10^{-04}$	1,990000000000001	1000
400	$4{,}307163317619962 \cdot 10^{-04}$	1,000000000000000	1000
900	$2{,}791228361303801 \cdot 10^{-05}$	1,060000000000000	1000
1600	$2{,}489158981260443 \cdot 10^{-05}$	1,350000000000000	1000
2500	$3{,}544701599452329 \cdot 10^{-05}$	1,540000000000000	1000
3600	$2{,}548697922521707 \cdot 10^{-05}$	1,660000000000001	1000
4900	$1{,}762003054063865 \cdot 10^{-05}$	1,740000000000001	1000
6400	$3{,}637439351604375 \cdot 10^{-05}$	1,800000000000001	1000
8100	$3{,}640402110359138 \cdot 10^{-05}$	1,840000000000001	1000
10000	$4{,}121331787828775 \cdot 10^{-05}$	1,870000000000001	1000
1000000	$1{,}523104812887002 \cdot 10^{+02}$	1,990000000000001	1000

Tabla 46: Error del segundo ejemplo usando SOR con w óptimo

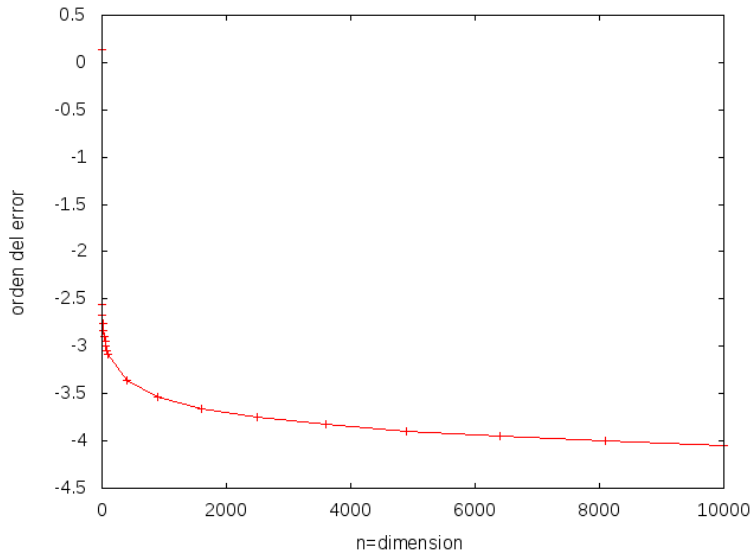


Figura 24: Error del segundo ejemplo en escala logarítmica usando SOR

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos concluir que para dicho ejemplo, el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar pero mejora a Gauss-Seidel y Jacobi a partir de dimensión 900.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios					
Sistema	n	Error	Iteraciones	w óptimo	$\kappa(A)$
1	100	$2{,}830150304072656 \cdot 10^{-14}$	2	1,000000000000000	1,000000000000586
2	100	$4{,}778565891973270 \cdot 10^{-13}$	45	1,450000000000000	8,889579542351672
3	100	$4{,}870803693375212 \cdot 10^{-13}$	44	1,440000000000000	$1{,}045321894829862 \cdot 10$
4	100	$5{,}836252517001331 \cdot 10^{-13}$	52	1,500000000000000	$1{,}373886821611351 \cdot 10$
5	100	$5{,}139662569579361 \cdot 10^{-13}$	50	1,490000000000000	$1{,}245960228492898 \cdot 10$
6	100	$5{,}284403536083759 \cdot 10^{-13}$	46	1,440000000000000	$1{,}130441074930559 \cdot 10$
7	100	$5{,}594875261619472 \cdot 10^{-13}$	44	1,430000000000000	$1{,}018530744653610 \cdot 10$
8	100	$4{,}577103763266235 \cdot 10^{-13}$	37	1,380000000000000	7,673291683578189
9	100	$5{,}273910491195088 \cdot 10^{-13}$	47	1,470000000000000	$1{,}049783566225507 \cdot 10$
10	100	$5{,}215095144171975 \cdot 10^{-13}$	50	1,490000000000000	$1{,}203512121483044 \cdot 10$

Tabla 47: Error de 10 sistema aleatorios usando SOR

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anterior, exceptuando en el primer sistema que mejora los resultados, podemos concluir que el método nos proprciona datos similares a Crout, Doolittle, QR Givens, Gauss-Seidel y Jacobi pero que para el resto de métodos los mejora.

- Minimización:
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1					
n	Error	Iteraciones	n	Error	Iteraciones
1	0,000000000000000	1	20	$1{,}758640885322255 \cdot 10^{-01}$	1000
2	$4{,}556057734417251 \cdot 10^{-07}$	6	30	$1{,}842638516299868 \cdot 10^{-01}$	1000
3	$3{,}390486105056601 \cdot 10^{-04}$	840	40	$1{,}886220746410360 \cdot 10^{-01}$	1000
4	$7{,}073419237932407 \cdot 10^{-02}$	1000	50	$1{,}910610592912000 \cdot 10^{-01}$	1000
5	$1{,}162706460448722 \cdot 10^{-01}$	1000	60	$1{,}926346022626400 \cdot 10^{-01}$	1000
6	$1{,}234522780958392 \cdot 10^{-01}$	1000	70	$1{,}937859302034717 \cdot 10^{-01}$	1000
7	$1{,}384912882366800 \cdot 10^{-01}$	1000	80	$1{,}947040581491317 \cdot 10^{-01}$	1000
8	$1{,}395370908736212 \cdot 10^{-01}$	1000	90	$1{,}955232766546435 \cdot 10^{-01}$	1000
9	$1{,}503094810650894 \cdot 10^{-01}$	1000	100	$1{,}962981943011337 \cdot 10^{-01}$	1000
10	$1{,}520468967530476 \cdot 10^{-01}$	1000	1000	$2{,}020575718039881 \cdot 10^{-01}$	1000

Tabla 48: Error del primer ejemplo usando Minimización

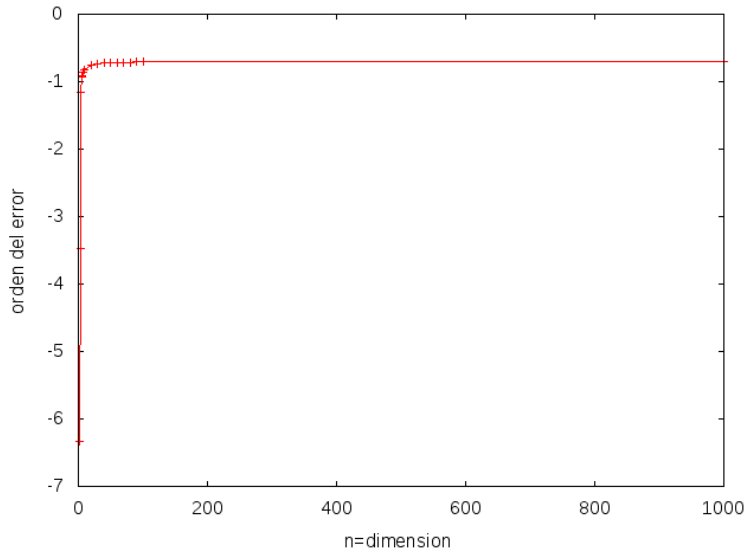


Figura 25: Error del primer ejemplo en escala logarítmica usando Minimización

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión hasta llegar a dimensión 4 donde se queda estable. Podemos observar que el método no nos da un buen resultado pero aún los podriamos dar por aceptables. Comparándolo con los métodos anteriores, vemos que pasa exactamente lo mismo que con el método de Gauss-Seidel con la diferencia que para dimensión 2 es el que peor resultado nos ha dado, a pesar de ser "bueno.[el] resultado que nos proporciona este método.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2					
n^2	Error	Iteraciones	n^2	Error	Iteraciones
1	$3{,}568515041634690 \cdot 10^{-03}$	1	400	$1{,}527525639196153 \cdot 10^{-04}$	1000
4	$2{,}755414257401030 \cdot 10^{-03}$	19	900	$1{,}875894526744836 \cdot 10^{-01}$	1000
9	$2{,}157474053920512 \cdot 10^{-03}$	44	1600	2,279389789265752	1000
16	$1{,}757791822006651 \cdot 10^{-03}$	71	2500	8,069013546332014	1000
25	$1{,}478231945162128 \cdot 10^{-03}$	105	3600	$1{,}715217388598009 \cdot 10$	1000
36	$1{,}273640347533151 \cdot 10^{-03}$	144	4900	$2{,}832334640788885 \cdot 10$	1000
49	$1{,}117623840437085 \cdot 10^{-03}$	189	6400	$4{,}061386751411957 \cdot 10$	1000
64	$9{,}949100271409261 \cdot 10^{-04}$	238	8100	$5{,}343660193745837 \cdot 10$	1000
81	$8{,}959478470370304 \cdot 10^{-04}$	294	10000	$6{,}648311972810332 \cdot 10$	1000
100	$8{,}145543275336171 \cdot 10^{-04}$	355			

Tabla 49: Error del segundo ejemplo usando Minimización

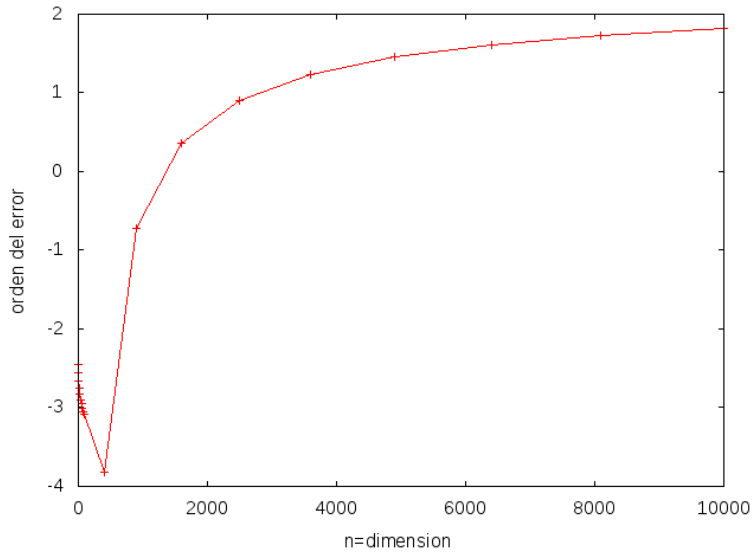


Figura 26: Error del segundo ejemplo en escala logarítmica usando Minimización

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión hasta llegar a dimensión 1600 donde empieza a empeorar. Podemos concluir que para dicho ejemplo, el método, hasta dimensión 900 es aceptable y para dimensión 1600 sigue siéndolo pero menos. Sin embargo, a partir de dimensión 1600, estos órdenes del error, son ya preocupantes y no aceptables por lo grandes que empiezan a ser. Con esto podemos concluir que este método empieza a fallar tanto que no podemos aceptar la solución dada por el sistema. Al igual que el primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión, hasta alcanzar la dimensión 900 pero, en contra del primer ejemplo, menos aconsejable conforme aumenta la dimensión a partir de dimensión 1600. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar hasta dimensión 900 pero los empeora a partir de dimensión 1600 aunque tiene precisión similar a Gauss-Seidel y Jacobi.

Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de sistemas aleatorios				
Sistema	n	Error	Iteraciones	$\kappa(A)$
1	100	$1{,}006150137588847 \cdot 10^{-11}$	1	1,000000000000586
2	100	$3{,}298917018263712 \cdot 10^{-07}$	16	1,952914610185349
3	100	$7{,}649846582252880 \cdot 10^{-07}$	15	2,257658263009223
4	100	$9{,}719074444175344 \cdot 10^{-07}$	15	2,192231996858926
5	100	$6{,}683719670672270 \cdot 10^{-07}$	22	5,590121445829727
6	100	$4{,}770296795683866 \cdot 10^{-07}$	25	7,467019443761775
7	100	$7{,}600706112176692 \cdot 10^{-07}$	29	$1{,}363225455326422 \cdot 10$
8	100	$8{,}393135803398785 \cdot 10^{-07}$	25	$1{,}047294431855527 \cdot 10$
9	100	$3{,}689820010552070 \cdot 10^{-07}$	21	4,148696728446243
10	100	$5{,}380555332426311 \cdot 10^{-07}$	25	$1{,}026573806329786 \cdot 10$

Tabla 50: Error de 10 sistema aleatorios usando Minimización

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anterior, vemos que el método nos proporciona peores datos que el resto.

- Gradiente conjugado:
  Aquí vamos a cambiar la condición inicial x por el vector donde cada componente vale 0.
  En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1					
n	Error	Iteraciones	n	Error	Iteraciones
1	0,000000000000000	1	20	$1{,}074408096138027 \cdot 10^{-04}$	19
2	$4{,}705040823708615 \cdot 10^{15}$	2	30	$1{,}166242361102504 \cdot 10^{-04}$	23
3	$6{,}960574574405189 \cdot 10^{13}$	3	40	$3{,}069107304188244 \cdot 10^{-04}$	22
4	$1{,}352021376164558 \cdot 10^{12}$	5	50	$1{,}788047476559421 \cdot 10^{-04}$	28
5	$7{,}199076742080798 \cdot 10^{12}$	7	60	$3{,}092498731559510 \cdot 10^{-04}$	29
6	$4{,}222677513233497 \cdot 10^{10}$	10	70	$1{,}562333452586264 \cdot 10^{-04}$	36
7	$1{,}812082486813386 \cdot 10^{-04}$	9	80	$2{,}306302276351738 \cdot 10^{-04}$	34
8	$3{,}466431826041678 \cdot 10^{-05}$	13	90	$3{,}193729744393001 \cdot 10^{-04}$	33
9	$8{,}578988628576195 \cdot 10^{-05}$	13	100	$1{,}503886404778573 \cdot 10^{-04}$	42
10	$1{,}602553144942379 \cdot 10^{-04}$	13	1000	$5{,}105195490293124 \cdot 10^{-04}$	84

Tabla 51: Error del primer ejemplo usando Gradiente Conjugado

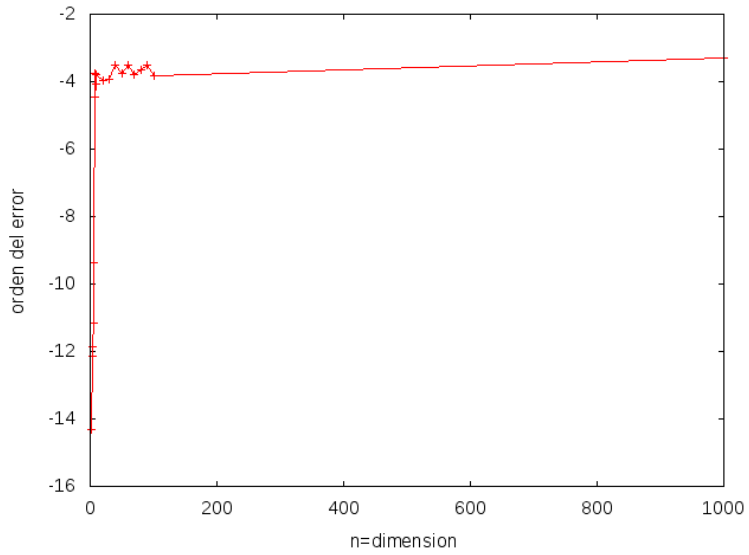


Figura 27: Error del primer ejemplo en escala logarítmica usando Gradiente Conjugado

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada pierde precisión. Podemos observar que el método es bueno para cualquier dimensión aunque con a partir de dimensión 20 los resultados empiezar a empeorar pero aún los podriamos dar por aceptables. Comparándolo con los métodos anteriores, vemos que es el mejor método visto hasta ahora (tanto directos como iterativos), sobretodo para dimensión grande.
Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2					
n^2	Error	Iteraciones	n^2	Error	Iteraciones
1	$3{,}568515041634690 \cdot 10^{-03}$	1	400	$4{,}307202390722196 \cdot 10^{-04}$	83
4	$2{,}755735094107417 \cdot 10^{-03}$	3	900	$2{,}920293729433868 \cdot 10^{-04}$	123
9	$2{,}158044791559726 \cdot 10^{-03}$	5	1600	$2{,}208714853580711 \cdot 10^{-04}$	163
16	$1{,}758522566483584 \cdot 10^{-03}$	9	2500	$1{,}775894304169555 \cdot 10^{-04}$	204
25	$1{,}479488029214253 \cdot 10^{-03}$	13	3600	$1{,}484883940223662 \cdot 10^{-04}$	243
36	$1{,}275238805385523 \cdot 10^{-03}$	19	4900	$1{,}275807955016151 \cdot 10^{-04}$	283
49	$1{,}119809922150506 \cdot 10^{-03}$	25	6400	$1{,}118336627728327 \cdot 10^{-04}$	323
64	$9{,}977841775990082 \cdot 10^{-04}$	31	8100	$9{,}954644558721323 \cdot 10^{-05}$	363
81	$8{,}995373727227325 \cdot 10^{-04}$	36	10000	$8{,}969178239819791 \cdot 10^{-05}$	403
100	$8{,}187854975012863 \cdot 10^{-04}$	40			

Tabla 52: Error del segundo ejemplo usando Gradiente Conjugado

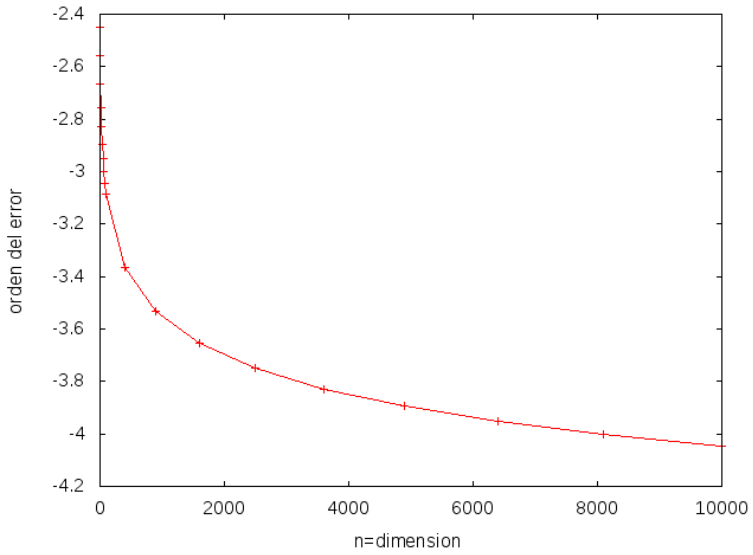


Figura 28: Error del segundo ejemplo en escala logarítmica usando Gradiente Conjugado

Mirando la tabla podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Podemos concluir que para dicho ejemplo, el método, para cualquier dimensión, es aceptable. En contra del primer ejemplo, este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos anteriores, vemos que el método funciona con precisión similar pero mejora a Gauss-Seidel y Jacobi a partir de dimensión 900.
Finalmente, para los sistemas aleatorios obtenemos:

Ejemplos de istemas aleatorios				
Sistema	n	Error	Iteraciones	$\kappa(A)$
1	100	$2{,}714890463506517 \cdot 10^{-13}$	2	1,000000000000586
2	100	$3{,}619739886471786 \cdot 10^{-13}$	15	8,889579542351672
3	100	$3{,}928787976625127 \cdot 10^{-13}$	15	$1{,}045321894829862 \cdot 10$
4	100	$4{,}324112752161946 \cdot 10^{-13}$	15	$1{,}373886821611351 \cdot 10$
5	100	$3{,}118679039926452 \cdot 10^{-13}$	15	$1{,}245960228492898 \cdot 10$
6	100	$7{,}328043491084560 \cdot 10^{-13}$	16	$1{,}130441074930559 \cdot 10$
7	100	$8{,}103218865785364 \cdot 10^{-13}$	14	$1{,}018530744653610 \cdot 10$
8	100	$3{,}964344898155795 \cdot 10^{-13}$	13	7,673291683578189
9	100	$5{,}275525569606507 \cdot 10^{-13}$	14	$1{,}049783566225507 \cdot 10$
10	100	$4{,}813870683219412 \cdot 10^{-13}$	14	$1{,}203512121483044 \cdot 10$

Tabla 53: Error de 10 sistema aleatorios usando Gradiente Conjugado

Se puede observar como para cualquier sistema, el método nos proporciona buenos datos independientemente del número de condición de la matriz. Comparándolo con los anterior, exceptuando en el primer sistema que mejora los resultados, podemos concluir que el método nos proprciona datos similares a Crout, Doolittle, QR Givens, Gauss-Seidel, Jacobi y SOR pero que para el resto de métodos los mejora.

2.5. Tiempo de ejecución y cálculo de operaciones

En este apartado vamos a ver el tiempo de ejecución y calcularemos el número de operaciones que es necesario hacer de cada método visto, dependiendo de la dimensión de nuestro sistema. Esto será útil a la hora de comparar métodos porque si uno tiene una precisión similar a otro pero requiere de más operaciones y tiempo de ejecución que otro pues a la hora de elegir siempre nos quedaríamos con el segundo.
Si n indica la dimensión del sistema, el número de operaciones de los métodos directos son:

Método	Sumas	Restas	Producto	División
Gauss	$\frac{n^2-n}{2}$	$\frac{n^3+2n-3}{3}$	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
Gauss con pivotaje	$\frac{n^2-n}{2}$	$\frac{n^3+2n-3}{3}$	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
Gauss con pivotaje total	$\frac{n^2-n}{2}$	$\frac{n^3+2n-3}{3}$	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
LU	n^2-n	$\frac{2n^3-3n^2+13n-12}{6}$	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
LU con pivotaje	n^2-n	$\frac{2n^3-3n^2+13n-12}{6}$	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
Doolittle	$\frac{2n^3+3n^2-5n}{6}$	n^2+2n-2	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
Crout	$\frac{2n^3+3n^2-5n}{6}$	n^2+2n-2	$\frac{2n^3+3n^2-5n}{6}$	$\frac{n^2+n}{2}$
Householder	$\frac{4n^4-7n^3+8n^2-5n}{2}$	n^2+n-2	$\frac{4n^4-3n^3+4n^2-5n}{2}$	n^2
Givens	$\frac{2n^3+3n^2-3n}{2}$	n^3-n^2+n-1	$\frac{8n^3-n^2-5n}{2}$	n^2

Tabla 54: Número de operaciones (suma, resta, producto y división) de los métodos directos

En cambio, el número de operaciones de los métodos iterativos y de Krylov tienen dos partes. La primera de ellas, escrito en azul, indica el número de operaciones obligatorias que cal hacer mientras que la segunda, escrito en negro, indica el número de operaciones que se realizan por iteración. Todo ello se resume en la siguiente tabla:

Método	Sumas	Restas	Producto	División
Jacobi	$\mathbf{0}+(n+1)$	$(\mathbf{n^2-n})+(n^2-n)$	$(\mathbf{n^2-n})+n^2$	$\mathbf{n}+n$
Gauss-Seidel	$\mathbf{0}+(n+1)$	$(\mathbf{n^2-n})+(n^2-n)$	$(\mathbf{n^2-n})+n^2$	$\mathbf{n}+n$
SOR	$\mathbf{n}+(2n+1)$	$\mathbf{n^2}+(n^2+2n)$	$(\mathbf{n^2+n})+(n^2+2n)$	$\mathbf{n}+n$
Minimización	$\mathbf{n^2}+(2n^2+2n)$	$\mathbf{0}+n$	$\mathbf{n^2}+(2n^2+3n)$	$\mathbf{0}+1$
Gradiente conjugado	$\mathbf{n}+(n^2+4n+1)$	$\mathbf{n^2}+n$	$(\mathbf{n^2+n})+(n^2+5n)$	$\mathbf{0}+2$
Gradiente biconjugado	$\mathbf{n}+(2n^2+6n+1)$	$\mathbf{n^2}+2n$	$(\mathbf{n^2+n})+(2n^2+8n)$	$\mathbf{0}+2$

Tabla 55: Número de operaciones (suma, resta, producto y división) de los métodos iterativos y Krylov

El número total de operaciones de los métodos directos es:

Método	Total
Gauss	$\frac{4n^3+9n^2-n-6}{6}$
Gauss con pivotaje	$\frac{4n^3+9n^2-n-6}{6}$
Gauss con pivotaje total	$\frac{4n^3+9n^2-n-6}{6}$
LU	$\frac{4n^3+9n^2+5n-12}{6}$
LU con pivotaje	$\frac{4n^3+9n^2+5n-12}{6}$
LU Doolittle	$\frac{4n^3+15n^2+5n-12}{6}$
LU Crout	$\frac{4n^3+15n^2+5n-12}{6}$
QR Householder	$4n^4 - 5n^3 + 8n^2 - 4n - 2$
QR Givens	$6n^3 + n^2 - 3n - 1$

Tabla 56: Número de operaciones totales de los métodos directos

(los iterativos sería lo azul más lo negro multiplicado por el número de iteraciones en las que se realizan que al desconocer dicho número no lo pondremos). Veamos ahora el tiempo de ejecución de cada método:

- Gauss:

	Gauss		Gauss con pivotaje		Gauss con pivotaje total	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,490s	1,370s	1,229s	2,676s	1,110s	0,682s
2	0,261s	0,489s	0,237s	0,531s	0,378s	0,469s
3	0,261s	0,333s	0,293s	1,686s	0,506s	0,277s
4	0,301s	0,353s	0,253s	0,831s	0,350s	0,317s
5	0,261s	0,361s	0,269s	0,551s	0,381s	0,333s
6	0,285s	0,381s	0,341s	0,471s	0,309s	0,373s
7	0,302s	0,349s	0,245s	0,779s	0,325s	0,397s
8	0,293s	0,365s	0,301s	1,279s	0,366s	0,390s
9	0,293s	0,389s	0,277s	1,266s	0,301s	0,313s
10	1,597s	0,877s	1,165s	0,870s	1,877s	1,485s
20	1,109s	1,002s	1,125s	1,346s	0,933s	1,173s
30	0,909s	0,881s	0,829s	2,819s	0,781s	2,858s
40	0,814s	1,248s	0,813s	12,524s	1,157s	12,407s
50	0,765s	2,133s	0,741s	41,318s	0,749s	45,152s
60	0,754s	4,327s	0,733s	2m 1,358s	0,721s	2m 14,339s
70	0,757s	8,108s	0,837s	5m 4,074s	0,737s	5m 32,240s
80	0,761s	18,379s	0,781s	11m 18,598s	1,429s	12m 24,807s
90	1,033s	26,402s	0,825s	22m 52,789s	0,901s	25m 1,211s
100	2,197s	43,385s	1,074s	43m 5,039s	1,123s	47m 13,596s
1000	3,523s	–	2,058s	–	4,277s	–

Tabla 57: Tiempo ejecución de los métodos de Gauss

- Descomposición LU:

	LU		LU con pivotaje	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,762s	0,636s	0,958s	0,938s
2	0,405s	0,475s	0,286s	0,457s
3	0,269s	0,334s	0,253s	0,382s
4	0,277s	0,343s	0,238s	0,301s
5	0,293s	0,319s	0,278s	0,365s
6	0,301s	0,335s	0,261s	0,357s
7	0,341s	0,502s	0,445s	0,341s
8	0,317s	0,407s	0,357s	0,325s
9	0,317s	0,679s	0,302s	0,333s
10	1,342s	0,967s	1,173s	1,022s
20	0,933s	0,870s	0,790s	0,821s
30	1,349s	1,000s	1,317s	0,937s
40	1,501s	1,540s	0,758s	1,694s
50	1,093s	2,717s	0,757s	3,172s
60	0,903s	5,803s	0,757s	5,707s
70	0,914s	10,572s	0,733s	12,139s
80	0,923s	19,484s	0,826s	21,194s
90	1,182s	33,260s	0,721s	36,647s
100	1,838s	55,820s	1,186s	1m 0,159s
1000	5,123s	–	1,987s	–

Tabla 58: Tiempo ejecución de la descomposición LU

- Variantes descomposición LU:

	Crout		Doolittle	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,906s	2,614s	6,249s	1,922s
2	0,293s	0,642s	0,285s	0,482s
3	0,277s	0,533s	0,269s	0,597s
4	0,309s	0,445s	0,262s	2,318s
5	0,261s	0,341s	0,237s	1,637s
6	0,301s	0,405s	0,357s	0,853s
7	0,245s	0,361s	0,293s	2,602s
8	0,325s	0,373s	0,333s	0,717s
9	0,317s	0,338s	0,325s	8,850s
10	1,765s	1,230s	1,261s	2,518s
20	0,894s	1,133s	0,805s	2,114s
30	0,817s	4,242s	0,834s	4,495s
40	0,813s	23,664s	0,761s	22,588s
50	0,762s	1m 35,428s	0,717s	1m 35,220s
60	0,846s	5m 22,266s	0,765s	5m 24,012s
70	0,801s	16m 12,324s	0,753s	16m 48,086s
80	0,793s	53m 16,895s	0,945s	53m 8,658s
90	0,981s	140m 51,101s	1,013s	139m 57,133s
100	1,021s	307m 10,617s	1,145s	304m58,907s
1000	6,121s	–	6,023s	–

Tabla 59: Tiempo ejecución de las variantes de la descomposición LU

- Descomposición QR:

	Householder		Givens	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,730s	1,122s	0,506s	1,381s
2	0,285s	0,349s	0,333s	0,365s
3	0,293s	0,293s	0,253s	0,335s
4	0,285s	0,313s	0,301s	0,335s
5	0,309s	0,441s	0,293s	0,363s
6	0,261s	0,442s	0,405s	0,367s
7	0,269s	0,477s	0,277s	0,335s
8	0,269s	0,937s	0,285s	0,351s
9	0,301s	1,209s	0,277s	0,630s
10	3,261s	2,781s	1,589s	1,614s
20	1,349s	14m 55,905s	1,141s	17,151s
30	1,433s	480m 24,441s	0,925s	4m 24,533s
40	0,989s	–	1,697s	32m 30,434s
50	1,005s	–	0,885s	151m 28,354s
60	1,137s	–	0,857s	540m 57,750s
70	1,473s	–	1,293s	–
80	1,585s	–	1,013s	–
90	1,941s	–	1,233s	–
100	2,789s	–	1,897s	–
1000	744m 46,953s	–	13m 59,045s	–

Tabla 60: Tiempo ejecución de la descomposición QR

- Iterativos:

	Gauss-Seidel		Jacobi		SOR	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	6,954s	–	1,086s	–	0,682s	–
2	0,373s	0,842s	0,341s	1,082s	0,337s	0,607s
3	0,253s	0,453s	0,365s	0,828s	0,761s	1,667s
4	0,349s	0,437s	0,301s	0,325s	0,328s	0,654s
5	0,237s	0,341s	0,269s	0,293s	0,305s	0,991s
6	0,437s	0,301s	0,237s	0,317s	0,396s	1,175s
7	0,381s	0,301s	0,277s	0,309s	0,381s	1,247s
8	0,557s	0,277s	0,285s	0,253s	0,921s	2,394s
9	0,461s	0,341s	0,285s	0,297s	0,429s	1,062s
10	1,081s	1,089s	1,101s	1,097s	1,397s	0,909s
20	0,985s	0,765s	1,125s	0,793s	1,270s	1,377s
30	1,653s	0,761s	1,321s	0,857s	3,318s	4,549s
40	0,793s	0,770s	0,969s	1,185s	2,504s	16,594s
50	0,921s	0,957s	0,849s	0,898s	2,978s	51,447s
60	0,885s	1,157s	0,845s	1,077s	4,078s	1m 44,671s
70	1,009s	1,037s	0,841s	1,089s	5,270s	3m 14,532s
80	0,942s	2,242s	0,766s	1,050s	6,602s	5m 37,230s
90	1,018s	1,363s	0,802s	1,131s	6,962s	9m 8,864s
100	1,462s	1,702s	1,534s	1,674s	8,506s	13m 56,830s
1000	8,351s	38,070s	2,071s	31,639s	11m 42,055s	–

Tabla 61: Tiempo ejecución de los métodos iterativos

- Krylov:

	Minimitzación		Gradiente conjugado	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,545s	0,962s	0,762s	0,834s
2	0,397s	0,797s	0,573s	0,429s
3	0,245s	0,557s	0,589s	0,405s
4	0,317s	0,581s	0,277s	0,405s
5	0,381s	0,678s	0,269s	0,397s
6	0,277s	0,669s	0,278s	0,421s
7	0,273s	0,833s	0,293s	0,533s
8	0,305s	0,512s	0,429s	0,373s
9	0,681s	0,469s	0,333s	0,377s
10	1,105s	1,149s	1,397s	1,061s
20	0,888s	4,529s	0,797s	0,941s
30	0,841s	23,009s	0,821s	1,610s
40	0,864s	1m 17,482s	0,877s	4,206s
50	0,821s	3m 16,441s	1,285s	10,664s
60	0,869s	8m 32,864s	0,830s	22,614s
70	0,881s	24m 44,948s	0,845s	50,427s
80	0,926s	48m 1,351s	0,841s	1m 33,204s
90	0,934s	73m 2,537s	0,797s	2m 44,994s
100	1,798s	115m 12,833s	1,319s	4m 37,084s
1000	29,478s	–	2,120s	–

Tabla 62: Tiempo ejecución de los métodos de Krylov

2.6. Error iterativo y error de discretización

Podemos calcular el error cometido entre dos iterados de cualquiera de los métodos iterativos vistos. Este error dependerá no solamente del método empleado sino también de la dimensión y de la condición inicial que le demos. Dicho error se puede calcular, independientemente del método, dimensión o condición inicial por:

$$x^{(k+1)} - x^{(k)} = B \cdot x^{(k)} + c - B \cdot x^{(k-1)} - c = B \cdot x^{(k)} - B \cdot x^{(k-1)} = B \cdot (x^{(k)} - x^{(k-1)})$$

Reiterando el proceso obtenemos:

$$x^{(k+1)} - x^{(k)} = B^k \cdot (x^{(1)} - x^{(0)})$$

Tomando una norma tenemos que:

$$||x^{(k+1)} - x^{(k)}|| \leq ||B||^k \cdot ||x^{(1)} - x^{(0)}||$$

En la tabla siguiente recogemos, en función de k, lo que nos da $||B||^k \cdot ||x^{(1)} - x^{(0)}||$ con los métodos de Jacobi y Gauss-Seidel en cada uno de nuestros ejemplos.

	Gauss-Seidel	
n	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	$2{,}000000000000000 \cdot 10^{-01}$	0,000000000000000
2	$2{,}000000000000002 \cdot 10^{-01}$	1,250000000000000
3	$3{,}021825396825393 \cdot 10^{-01}$	1,451388888888889
4	$3{,}135251322751323 \cdot 10^{-01}$	1,530787037037037
5	$3{,}977019164147139 \cdot 10^{-01}$	1,583592509920635
6	$4{,}085271737038446 \cdot 10^{-01}$	1,629347484016755
7	$4{,}785760069811277 \cdot 10^{-01}$	1,663811108459763
8	$4{,}882932249425731 \cdot 10^{-01}$	1,690426400194800
9	$5{,}481117366608056 \cdot 10^{-01}$	1,711403185282192
10	$5{,}568343565938323 \cdot 10^{-01}$	1,728218128203289
20	$8{,}026226535301753 \cdot 10^{-01}$	1,826974873126584
30	$9{,}655662820066743 \cdot 10^{-01}$	1,868076578796355
40	1,087870848277010	1,891693008597025
50	1,185895466452298	1,907449100330592
60	1,267736024816086	1,918914960540120
70	1,338003324762765	1,927596904336200
80	1,399577314857768	1,934473737226237
90	1,454378989183802	1,940088176318063
100	1,503754445690018	1,944753720594761
1000	2,627401189252652	1,993972491545839

Tabla 63: Error iterativo Gauss-Seidel ejemplo 1 con norma 1

	Gauss-Seidel	
n	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	$2{,}000000000000000 \cdot 10^{-01}$	0,000000000000000
2	$2{,}000000000000000 \cdot 10^{-01}$	$9{,}013878188659973 \cdot 10^{-01}$
3	$2{,}508904054120857 \cdot 10^{-01}$	$9{,}990009796325179 \cdot 10^{-01}$
4	$2{,}511466701641935 \cdot 10^{-01}$	1,000088525965086
5	$2{,}792411866926854 \cdot 10^{-01}$	1,000291094252221
6	$2{,}794509376275466 \cdot 10^{-01}$	1,000512191288073
7	$2{,}982114395631588 \cdot 10^{-01}$	1,000739479007331
8	$2{,}983697153050922 \cdot 10^{-01}$	1,000959021598935
9	$3{,}120700617124423 \cdot 10^{-01}$	1,001164645901008
10	$3{,}121919401837651 \cdot 10^{-01}$	1,001354307050643
20	$3{,}493572844658004 \cdot 10^{-01}$	1,002556994509109
30	$3{,}666989012725747 \cdot 10^{-01}$	1,003115316355799
40	$3{,}770688698340312 \cdot 10^{-01}$	1,003429248142905
50	$3{,}840778416005514 \cdot 10^{-01}$	1,003629079059051
60	$3{,}891801996444038 \cdot 10^{-01}$	1,003767044971887
70	$3{,}930854509368404 \cdot 10^{-01}$	1,003867865144454
80	$3{,}961846554753083 \cdot 10^{-01}$	1,003944687969902
90	$3{,}987125754781198 \cdot 10^{-01}$	1,004005132302866
100	$4{,}008193731830081 \cdot 10^{-01}$	1,004053910361670
1000	$4{,}230633281580370 \cdot 10^{-01}$	1,004460683668598

Tabla 64: Error iterativo Gauss-Seidel ejemplo 1 con norma 2

	Gauss-Seidel	
n	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	$2{,}000000000000000 \cdot 10^{-01}$	0,000000000000000
2	$4{,}999999999999998 \cdot 10^{-01}$	1,500000000000000
3	1,130952380952380	3,166666666666667
4	1,647619047619047	4,916666666666667
5	2,556768707482993	6,716666666666667
6	3,242086167800453	8,550000000000001
7	4,361019002642383	$1{,}040714285714286 \cdot 10$
8	5,186596203219587	$1{,}228214285714286 \cdot 10$
9	6,474342538081307	$1{,}417103174603175 \cdot 10$
10	7,420477791716559	$1{,}607103174603175 \cdot 10$
20	$2{,}177623710089362 \cdot 10$	$3{,}540226034285632 \cdot 10$
30	$3{,}987602912110724 \cdot 10$	$5{,}500501286907961 \cdot 10$
40	$6{,}061365874133580 \cdot 10$	$7{,}472145696106362 \cdot 10$
50	$8{,}340155874095515 \cdot 10$	$9{,}450079466167060 \cdot 10$
60	$1{,}078712751724747 \cdot 10^{+02}$	$1{,}143201295870483 \cdot 10^{+02}$
70	$1{,}337689168936974 \cdot 10^{+02}$	$1{,}341671632423620 \cdot 10^{+02}$
80	$1{,}609084131128423 \cdot 10^{+02}$	$1{,}540345207210545 \cdot 10^{+02}$
90	$1{,}891473049251367 \cdot 10^{+02}$	$1{,}739174293971515 \cdot 10^{+02}$
100	$2{,}183728969770332 \cdot 10^{+02}$	$1{,}938126224823604 \cdot 10^{+02}$
1000	$4{,}206031466275197 \cdot 10^{+03}$	$1{,}991514529139451 \cdot 10^{+03}$

Tabla 65: Error iterativo Jacobi ejemplo 1 con norma 1

	Gauss-Seidel	
n	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	$2{,}000000000000000 \cdot 10^{-01}$	0,000000000000000
2	$3{,}605551275463987 \cdot 10^{-01}$	1,500000000000000
3	$6{,}736748982118342 \cdot 10^{-01}$	2,502477790831270
4	$8{,}489889945842465 \cdot 10^{-01}$	3,580891042607409
5	1,183701196287362	4,673829413483642
6	1,367775033954088	5,771337441841066
7	1,707526475027460	6,870691726735869
8	1,896635014185016	7,970907326173736
9	2,235596018653494	9,071562772737572
10	2,427562867731916	$1{,}017245633009062 \cdot 10$
20	5,031480044354033	$2{,}118399975580906 \cdot 10$
30	7,511171469848409	$3{,}219544535598050 \cdot 10$
40	9,874591012216129	$4{,}320647255669795 \cdot 10$
50	$1{,}213893770670851 \cdot 10$	$5{,}421726048081495 \cdot 10$
60	$1{,}431879034834739 \cdot 10$	$6{,}522790674787757 \cdot 10$
70	$1{,}642566005039744 \cdot 10$	$7{,}623846334805765 \cdot 10$
80	$1{,}846864296607453 \cdot 10$	$8{,}724895985221298 \cdot 10$
90	$2{,}045503052126865 \cdot 10$	$9{,}825941418584914 \cdot 10$
100	$2{,}239076355091995 \cdot 10$	$1{,}092698378120097 \cdot 10^{+02}$
1000	$1{,}348754438072891 \cdot 10^{+02}$	$1{,}100196728107869 \cdot 10^{+03}$

Tabla 66: Error iterativo Jacobi ejemplo 1 con norma 2

	Gauss-Seidel	
n^2	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	1,298757942860316	$2{,}500000000000000 \cdot 10^{-01}$
4	3,501362219806202	$4{,}062500000000000 \cdot 10^{-01}$
9	5,688095013190456	$4{,}667968750000000 \cdot 10^{-01}$
16	7,832195118849835	$4{,}885253906250000 \cdot 10^{-01}$
25	9,931471177405891	$4{,}960861206054688 \cdot 10^{-01}$
36	$1{,}199936746673787 \cdot 10$	$4{,}986753463745117 \cdot 10^{-01}$
49	$1{,}404672544476281 \cdot 10$	$4{,}995538592338562 \cdot 10^{-01}$
64	$1{,}608040754918038 \cdot 10$	$4{,}998502209782600 \cdot 10^{-01}$
81	$1{,}810458153436107 \cdot 10$	$4{,}999498239485547 \cdot 10^{-01}$
100	$2{,}012184003638558 \cdot 10$	$4{,}999832156900084 \cdot 10^{-01}$
400	$4{,}012644898871233 \cdot 10$	$4{,}999999997133098 \cdot 10^{-01}$
900	$6{,}002743483381581 \cdot 10$	$4{,}999999999999946 \cdot 10^{-01}$
1600	$7{,}989921049785283 \cdot 10$	$4{,}999999999999988 \cdot 10^{-01}$
2500	$9{,}975873848655849 \cdot 10$	$4{,}999999999999988 \cdot 10^{-01}$
3600	$1{,}196119830285605 \cdot 10^{+02}$	$4{,}999999999999988 \cdot 10^{-01}$
4900	$1{,}394615778059494 \cdot 10^{+02}$	$4{,}999999999999988 \cdot 10^{-01}$
6400	$1{,}593088654093134 \cdot 10^{+02}$	$4{,}999999999999988 \cdot 10^{-01}$
8100	$1{,}791546019678183 \cdot 10^{+02}$	$4{,}999999999999988 \cdot 10^{-01}$
10000	$1{,}989992457920750 \cdot 10^{+02}$	$4{,}999999999999988 \cdot 10^{-01}$

Tabla 67: Error iterativo Gauss-Seidel ejemplo 2 con norma 1

	Gauss-Seidel	
n^2	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	1,298757942860316	$2{,}500000000000000 \cdot 10^{-01}$
4	1,761302193420510	$3{,}259197361565169 \cdot 10^{-01}$
9	1,953719556436013	$3{,}755204976471128 \cdot 10^{-01}$
16	2,103869378633563	$4{,}079125367917462 \cdot 10^{-01}$
25	2,242474229520139	$4{,}297001041062515 \cdot 10^{-01}$
36	2,375305731645105	$4{,}448530915527644 \cdot 10^{-01}$
49	2,503299269416579	$4{,}557280203957152 \cdot 10^{-01}$
64	2,626707692008393	$4{,}637542368858561 \cdot 10^{-01}$
81	2,745780364342936	$4{,}698250313311607 \cdot 10^{-01}$
100	2,860805952221732	$4{,}745161005358021 \cdot 10^{-01}$
400	3,844050063066811	$4{,}923416943027735 \cdot 10^{-01}$
900	4,633405431455647	$4{,}963920244588511 \cdot 10^{-01}$
1600	5,309477107427061	$4{,}979124856897059 \cdot 10^{-01}$
2500	5,909798407703478	$4{,}986416933339083 \cdot 10^{-01}$
3600	6,455050436808495	–
4900	6,957988747693640	–
6400	7,427116660849469	–
8100	7,868431764044223	–
10000	8,286348840442088	–

Tabla 68: Error iterativo Gauss-Seidel ejemplo 2 con norma 2

NOTA: A partir de $n^2 = 3600$ no tenemos el calculo de $||B||$ pues calcular la inversa de $(D + L)$, su producto por U y finalmente hayar su radio espectral por el método de la potencia es muy costoso y el tiempo incrementa mucho. Aún asi, a la vista de los resultados podemos aproximar dicho número por $5 \cdot 10^{-01}$.

	Gauss-Seidel	
n^2	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	1,385341805717671	0,000000000000000
4	2,690244246784248	$5{,}000000000000000 \cdot 10^{-01}$
9	3,935147753992304	1,000000000000000
16	5,150527040362872	1,000000000000000
25	6,349703371367809	1,000000000000000
36	7,539110893863129	1,000000000000000
49	8,722195286742236	1,000000000000000
64	9,900958707523777	1,000000000000000
81	$1{,}107664122252404 \cdot 10$	1,000000000000000
100	$1{,}225005089913230 \cdot 10$	1,000000000000000
400	$2{,}392918356003327 \cdot 10$	1,000000000000000
900	$3{,}557522374141389 \cdot 10$	1,000000000000000
1600	$4{,}721216370434827 \cdot 10$	1,000000000000000
2500	$5{,}884531865941894 \cdot 10$	1,000000000000000
3600	$7{,}047653926338900 \cdot 10$	1,000000000000000
4900	$8{,}210663880413264 \cdot 10$	1,000000000000000
6400	$9{,}373603070095220 \cdot 10$	1,000000000000000
8100	$1{,}053649473521557 \cdot 10^{+02}$	1,000000000000000
10000	$1{,}169935294357102 \cdot 10^{+02}$	1,000000000000000

Tabla 69: Error iterativo Jacobi ejemplo 2 con norma 1

	Gauss-Seidel	
n^2	$\|\|x^{(1)} - x^{(0)}\|\|$	$\|\|B\|\|$
1	1,385341805717671	0,000000000000000
4	1,357136109100437	$5{,}000000000000000 \cdot 10^{-01}$
9	1,453605263111305	$7{,}071067811865476 \cdot 10^{-01}$
16	1,550450696707138	$8{,}090169943749450 \cdot 10^{-01}$
25	1,646469222931466	$8{,}660254037844359 \cdot 10^{-01}$
36	1,740397221931556	$9{,}009688679023884 \cdot 10^{-01}$
49	1,831628897308228	$9{,}238795325111379 \cdot 10^{-01}$
64	1,919967570187985	$9{,}396926207858083 \cdot 10^{-01}$
81	2,005426367855819	$9{,}510565162948673 \cdot 10^{-01}$
100	2,088117594298807	$9{,}594929736140653 \cdot 10^{-01}$
400	2,796551270754795	$9{,}888308262229635 \cdot 10^{-01}$
900	3,365634823840174	$9{,}948693233864648 \cdot 10^{-01}$
1600	3,853206152821011	$9{,}970658011734623 \cdot 10^{-01}$
2500	4,286289573026031	$9{,}981033287212323 \cdot 10^{-01}$
3600	4,679755494554978	–
4900	5,042771120198304	–
6400	5,381447501244461	–
8100	5,700095662805269	–
10000	6,001889730655477	–

Tabla 70: Error iterativo Jacobi ejemplo 2 con norma 2

Para el segundo ejemplo, hemos empleado un método (el de la fórmula centrada de la segunda derivada) que ya de por si tiene un error. Este error se le llama error de discretización. Vamos a ver cual es el error cometido por usar tal método: Haciendo Taylor a $f(x+h)$ y $f(x-h)$ y queda:

$$f(x+h) = f(x) + f'(x) \cdot h + \frac{f''(x)}{2} \cdot h^2 + \frac{f'''(x)}{3!} \cdot h^3 + \frac{f^{(IV)}(x)}{4!} \cdot h^4 + \cdots$$

$$f(x-h) = f(x) - f'(x) \cdot h + \frac{f''(x)}{2} \cdot h^2 - \frac{f'''(x)}{3!} \cdot h^3 + \frac{f^{(IV)}(x)}{4!} \cdot h^4 - \cdots$$

Sumando estas expresiones queda:

$$f(x+h)+f(x-h)=2\cdot f(x)+f''(x)\cdot h^2+\frac{f^{(IV)}(x)}{12}\cdot h^4+\cdots$$

Despejando $f''(x)$ queda:

$$f''(x)=\frac{f(x+h)+f(x-h)-2\cdot f(x)}{h^2}-\frac{f^{(IV)}(x)}{12}\cdot h^2-\cdots-2\cdot\frac{f^{(2k)}(x)}{(2k)!}\cdot h^{2k-2}-\cdots$$

Asi pues el error de discretización de la formula es

$$E(f,h)=\sum_{k=2}^{\infty}2\cdot\frac{f^{(2k)}(x)}{(2k)!}\cdot h^{2k-2}$$

Como h es pequeño, podemos encontrar un N tal que $h^{2(N+1)-2}=h^{2N}$ sea más pequeño que una cierta tolerancia y considerar $h^{2k-2}=0\ \forall k>N$, de esta manera:

$$E(f,h)=E(f,h,N)=\sum_{k=2}^{N}2\cdot\frac{f^{(2k)}(x)}{(2k)!}\cdot h^{2k-2}$$

Por último, si dicho h es muy pequeño, podemos quedarnos hasta $N=2$, es decir

$$E(f,h)=E(f,h,2)=\frac{f^{(IV)}(x)}{12}\cdot h^2$$

Observación 9. *Hemos observado que el ejemplo 2 obtiene mejores resultados conforme aumenta la dimensión del sistema. Esto se debe especialmente a que el aumento de la dimensión reduce el error de discretización*

2.7. Cota error y condición matriz

Como hemos comentado en los preliminares, es importante que una matriz éste bien concionada. Para ello la mejor manera de saberlo, es ver el número de condición de dicha matriz es próximo a uno o no. Es por ello que vamos a ver si nuestras matrices están bien condicionada o mal condicionada calculando su número de condición. Dichos valores se resume en la siguiente tabla:

$\kappa(A)$			
n	Ejemplo 1	Ejemplo 2	Iteraciones ejemplo 2
1	1,000000000000000	1,000000000000000	2
2	$2{,}700000000000001 \cdot 10$	3,000000000000000	2
3	$7{,}480000000000027 \cdot 10^{+02}$	9,000000000000000	3
4	$2{,}837499999999738 \cdot 10^{+04}$	$1{,}333333333333333 \cdot 10$	3
5	$9{,}436559999999363 \cdot 10^{+05}$	$2{,}076923076923077 \cdot 10$	3
6	$2{,}907027900294877 \cdot 10^{+07}$	$2{,}744827586206896 \cdot 10$	3
7	$9{,}851948898250020 \cdot 10^{+08}$	$3{,}726470588235295 \cdot 10$	3
8	$3{,}387279072526593 \cdot 10^{+10}$	$4{,}629522752497220 \cdot 10$	3
9	$1{,}099650911956081 \cdot 10^{+12}$	$5{,}847874842732892 \cdot 10$	3
10	$3{,}535378193923801 \cdot 10^{+13}$	$6{,}986337089651062 \cdot 10$	3
20	$1{,}425868147996027 \cdot 10^{+16}$	$2{,}584519983485457 \cdot 10^{+02}$	3
30	$2{,}319804158849641 \cdot 10^{+17}$	$5{,}649227415279706 \cdot 10^{+02}$	3
40	$1{,}101195860903429 \cdot 10^{+16}$	$9{,}892689755980344 \cdot 10^{+02}$	3
50	$6{,}006625651751126 \cdot 10^{+16}$	$1{,}531489776006641 \cdot 10^{+03}$	3
60	$4{,}436629740889831 \cdot 10^{+16}$	$2{,}191584903409613 \cdot 10^{+03}$	3
70	$4{,}210047101399366 \cdot 10^{+16}$	$2{,}969554273604909 \cdot 10^{+03}$	3
80	$6{,}302959861525204 \cdot 10^{+16}$	$3{,}865397850819386 \cdot 10^{+03}$	3
90	$7{,}820668763791368 \cdot 10^{+16}$	$4{,}879115617759055 \cdot 10^{+03}$	3
100	$1{,}705767274129921 \cdot 10^{+16}$	$6{,}010707565234711 \cdot 10^{+03}$	3
1000	$2{,}995861264820046 \cdot 10^{+21}$	–	3

Tabla 71: Número de condición de las matrices de los dos ejemplos vistos

En el primer ejemplo esta hecho por la propia definición mientras que el segundo esta hecho por Hager. Finalmente en el libro [3], en el problema 2.5 se puede encontrar una cota del error, Si A tiene un error δA, b tiene un error δb y x tiene error δx entonces:

$$\frac{||\delta x||}{||x||} = \frac{\mu(A)}{1 - \mu(A) \cdot \frac{||\delta A||}{||A||}} \cdot (\frac{||\delta b||}{||b||} + \frac{||\delta A||}{||A||})$$

donde $\mu(A)$ indica el número de condición de la matriz A. Aplicandolo a nuestros ejemplos obtenemos la tabla siguiente:

Ejemplo 1	
n	Cota del error
1	$2{,}168404344971009 \cdot 10^{-19}$
2	$6{,}412281420128557 \cdot 10^{-18}$
3	$1{,}984617425354024 \cdot 10^{-16}$
4	$8{,}330932195662100 \cdot 10^{-15}$
5	$3{,}032726958309911 \cdot 10^{-13}$
6	$1{,}013125781314707 \cdot 10^{-11}$
7	$3{,}694507458275623 \cdot 10^{-10}$
8	$1{,}358011355204538 \cdot 10^{-08}$
9	$4{,}687865893119895 \cdot 10^{-07}$
10	$1{,}595287882006114 \cdot 10^{-05}$
20	$3{,}996839543555821 \cdot 10$
30	$2{,}205099185614128 \cdot 10^{-01}$
40	$9{,}174205353230724 \cdot 10^{-02}$
50	$-1{,}635225862151859$
60	$1{,}657878738939191 \cdot 10^{-01}$
70	$1{,}125488966588198 \cdot 10^{-01}$
80	$5{,}864262312769951 \cdot 10^{-02}$
90	$2{,}912820602750451 \cdot 10^{-02}$
100	$1{,}034628792155191 \cdot 10^{-01}$
1000	$1{,}095116103743394 \cdot 10^{-02}$

Tabla 72: Cota del error de la matriz del primer ejemplo

Comparando la tabla con los errores obtenidos en el primer ejemplo, se puede observar como según el método elegido y la dimensión la cota del error nos sirve y en otros no.

Ejemplo 2	
n^2	Cota del error
1	$2{,}575909444807402 \cdot 10^{-03}$
4	$8{,}302565905529807 \cdot 10^{-03}$
9	$1{,}524897861481848 \cdot 10^{-02}$
16	$1{,}367021329219936 \cdot 10^{-02}$
25	$1{,}358512498335538 \cdot 10^{-02}$
36	$1{,}206015530369957 \cdot 10^{-02}$
49	$1{,}148633047473678 \cdot 10^{-02}$
64	$1{,}037561114094897 \cdot 10^{-02}$
81	$9{,}816290618775302 \cdot 10^{-03}$
100	$9{,}004269351476676 \cdot 10^{-03}$
400	$5{,}285050610865745 \cdot 10^{-03}$
900	$3{,}713709684859111 \cdot 10^{-03}$
1600	$2{,}859020716812064 \cdot 10^{-03}$
2500	$2{,}323218482296940 \cdot 10^{-03}$
3600	$1{,}956227741563917 \cdot 10^{-03}$
4900	$1{,}689230287271674 \cdot 10^{-03}$
6400	$1{,}486298246062285 \cdot 10^{-03}$
8100	$1{,}326860539131600 \cdot 10^{-03}$
10000	$1{,}198295879558048 \cdot 10^{-03}$

Tabla 73: Cota del error de la matriz del segundo ejemplo

Comparando la tabla con los errores obtenidos en el segundo ejemplo, se puede observar como según el método elegido y la dimensión la cota del error la mayoría de las veces nos funciona..

3. Métodos numéricos de calculo de valores propios

Tal y como hemos visto, a veces es necesario calcular el radio espectral de una matriz A ya sea para encontrar el párametro de relajación w de una matriz simétrica, definida positiva, real y tridiagonal a bloques como para comprobar la convergencia de un método iterativo.
En este capítulo veremos un método para encontrar el radio espectral de una matriz y una variante de dicho método para encontrar el resto de valores propios. Dicho método encuentra el vector propio del valor propio el radio espectral. Para encontrar tal valor propio asociado a este vector propio se usa el cociente de Rayleigh:

Definición 22.
Sea v un vector propio de A (conocido). Entonces el valor propio asociado a este vector propio v es:

$$\lambda = \frac{\bar{v}^t \cdot A \cdot v}{\bar{v}^t \cdot v}$$

Dicha igualdad se la conoce como cociente de Rayleigh

También veremos que la descomposición QR también nos sirve para calcular los valores propios de una matriz A fácilmente y veremos la descomposición en valores singulares (SVD). Finalmente, veremos otros métodos como Arnoldi y Lanczos el cual requerirán la siguiente definición:

Definición 23.
Una sucesión $\{f_0, f_1, \cdots, f_n\}$ de funciones reales continuas, definidas sobre un intervalo [a,b] es una sucesión de Sturm para $f = f_0$ sobre $[a, b]$ si verifica:

1. *f_0 es diferenciable en [a,b].*
2. *f_n no tiene ceros reales en [a,b].*
3. *Si $f_0(x) = 0$ entonces $f_1(x) \cdot f_0'(x) > 0$.*
4. *Si $f_i(x) = 0$ entonces $f_{i+1}(x) \cdot f_{i-1}(x) < 0$ para $1 \leq i \leq n-1$.*

3.1. Método de la potencia

Sea A una matriz $n \times n$ con valores propios λ_1, λ_2, λ_3, ..., λ_n tal que

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$$

El método de la potencia es un método que permite encontrar λ_1 partiendo de la idea siguiente:

$$\begin{cases} \text{Dado un } z_0 \in \Re^n \setminus 0 \\ z_{k+1} = A \cdot z_k = A^{k+1} \cdot z_0 \end{cases}$$

Supongamos que existe una base de vectores propios (si no se haría mediante una base de Jordan pero sería más laborioso de hacerlo). Sea v_1, v_2, v_3, ..., v_n vectores

propios de valor propio λ_1, λ_2, λ_3, ..., λ_n respectivamente. Así:
$z_0 = \alpha_1 \cdot v_1 + \alpha_2 \cdot v_2 + \ldots + \alpha_n \cdot v_n$
$z_1 = A \cdot z_0 = \alpha_1 \cdot A \cdot v_1 + \alpha_2 \cdot A \cdot v_2 + \ldots + \alpha_n \cdot A \cdot v_n = \alpha_1 \cdot \lambda_1 \cdot v_1 + \alpha_2 \cdot \lambda_2 \cdot v_2 + \ldots + \alpha_n \cdot \lambda_n \cdot v_n$
$z_2 = A \cdot z_1 = \alpha_1 \cdot \lambda_1 \cdot A \cdot v_1 + \alpha_2 \cdot \lambda_2 \cdot A \cdot v_2 + \ldots + \alpha_n \cdot \lambda_n \cdot A \cdot v_n = \alpha_1 \cdot (\lambda_1)^2 \cdot v_1 + \alpha_2 \cdot (\lambda_2)^2 \cdot v_2 + \ldots + \alpha_n \cdot (\lambda_n)^2 \cdot v_n$

$\vdots$

$z_k = A \cdot z_{k-1} = \alpha_1 \cdot (\lambda_1)^{k-1} \cdot A \cdot v_1 + \alpha_2 \cdot (\lambda_2)^{k-1} \cdot A \cdot v_2 + \ldots + \alpha_n \cdot (\lambda_n)^{k-1} \cdot A \cdot v_n = \alpha_1 \cdot (\lambda_1)^k \cdot v_1 + \alpha_2 \cdot (\lambda_2)^k \cdot v_2 + \ldots + \alpha_n \cdot (\lambda_n)^k \cdot v_n = (\lambda_1)^k \cdot [\alpha_1 \cdot v_1 + \sum_{j=2}^{n} (\alpha_j \cdot (\frac{\lambda_j}{\lambda_1})^k \cdot v_j)]$
Como $|\lambda_1| > |\lambda_j| \ \forall j \geq 2 \implies \frac{|\lambda_j|}{|\lambda_1|} \ \forall j \geq 2 \implies \frac{|\lambda_j|}{|\lambda_1|}^k \ \forall j \geq 2 \ \forall k \geq 1 \implies$
$\lim_{k\to\infty} \frac{|\lambda_j|}{|\lambda_1|}^k = 0 \ \forall j \geq 2 \implies \lim_{k\to\infty} [\alpha_1 \cdot v_1 + \sum_{j=2}^{n} (\alpha_j \cdot (\frac{\lambda_j}{\lambda_1})^k \cdot v_j)] = \alpha_1 \cdot v_1$
Así

$$\lim_{k\to\infty} z_k = \alpha_1 \cdot v_1 \cdot \lim_{k\to\infty} (\lambda_1)^k = \begin{cases} \alpha_1 \cdot v_1 \cdot \infty = \infty & \text{si } |\lambda_1| > 1 \\ \alpha_1 \cdot v_1 \cdot 1 = \alpha_1 \cdot v_1 \cdot 1 & \text{si } |\lambda_1| = 1 \\ \alpha_1 \cdot v_1 \cdot 0 = 0 & \text{si } |\lambda_1| < 1 \end{cases}$$

Con esto concluimos que el método de la potencia es:
Dado un $z_0 \in \Re^n \setminus 0$ definimos la siguiente recurrencia:

$$\begin{cases} y_k = \frac{z_k}{||z_k||} \\ z_{k+1} = A \cdot y_k \\ \lambda_k = y_k^t \cdot z_{k+1} \end{cases}$$

donde y_k indica el vector normalizado de z_k y λ_k es el cociente de Rayleigh.

3.2. Variantes del método de la potencia

3.2.1. Método de la potencia inversa

Si A tiene valores propios λ_1, λ_2, λ_3, $\cdots$, λ_n entonces A^{-1} tiene valores propios $\frac{1}{\lambda_1}$, $\frac{1}{\lambda_2}$, $\frac{1}{\lambda_3}$, $\cdots$, $\frac{1}{\lambda_n}$. Asi si

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_{n-1}| > |\lambda_n| \implies \frac{1}{|\lambda_1|} \leq \frac{1}{|\lambda_2|} \leq \frac{1}{|\lambda_3|} \leq \cdots \leq \frac{1}{|\lambda_{n-1}|} < \frac{1}{|\lambda_n|}$$

Asi aplicando el método de la potencia a A^{-1} obtenemos la recurrencia

$$\begin{cases} y_k = \frac{z_k}{||z_k||} \\ z_{k+1} = A^{-1} \cdot y_k \Leftrightarrow A \cdot z_{k+1} = y_k \\ \lambda_k = y_k^t \cdot z_{k+1} \end{cases}$$

que obtenemos como resultado $\frac{1}{|\lambda_n|}$.

3.2.2. Método de la potencia desplazada

Si A tiene valores propios λ_1, λ_2, λ_3, $\cdots$, λ_n entonces $A - \mu \cdot Id$, donde Id es la matriz identidad, tiene valores propios $\lambda_1 - \mu$, $\lambda_2 - \mu$, $\lambda_3 - \mu$, $\cdots$, $\lambda_n - \mu$. El objetivo de este método es encontrar el valor propio más cercano a μ. Para ello, hemos de pedir el valor propio de módulo minimo de $A - \mu \cdot Id$, equivalentemente el valor propio de módulo máximo de $(A - \mu \cdot Id)^{-1}$. Así pues hemos de aplicar el método de la potencia inversa a $A - \mu \cdot Id$ y obtenemos la recurrencia:

$$\begin{cases} y_k = \frac{z_k}{||z_k||} \\ z_{k+1} = (A - \mu \cdot Id)^{-1} \cdot y_k \Leftrightarrow (A - \mu \cdot Id) \cdot z_{k+1} = y_k \\ \lambda_k = y_k^t \cdot z_{k+1} \end{cases}$$

que obtenemos el inverso del valor propio de A más cercano a μ.

Resumen:
Si queremos encontrar los valores propios de una matriz A podemos hacer lo siguiente:

1. Método de la potencia con A para encontrar λ_1 y método de la potencia inversa con A para encontrar $\frac{1}{\lambda_n}$
2. Método de la potencia desplazada con $A - \lambda_1$ y con $A - \lambda_n$ para encontrar $\frac{1}{\lambda_2}$ y $\frac{1}{\lambda_{n-1}}$
3. Método de la potencia desplazada con $A - \lambda_2$ y con $A - \lambda_{n-1}$ para encontrar $\frac{1}{\lambda_3}$ y $\frac{1}{\lambda_{n-2}}$
   Y así sucesivamente, se repite el proceso hasta encontrarlos todos

3.2.3. Aplicación

Una de las principales empresas que usa dicho método es Google. Básicamente, lo usan para calcular el PageRank, número que se le asigna a una página web según su relevancia (del cual a la hora de buscar cualquier cosa en Google, éste te lo ordenadara según dicho valor de aquellos enlaces que tengan relación con tu búsqueda). Las páginas webs se enlazan entre ellas, formando así un grafo dirigido del cual cada arista de i a j indica que en la página i hay un enlace de la página j. El PageRank depende básicamente de la cantidad de páginas que te enlacen y de la importancia de dichas páginas. Tal y como dijimos en los preliminares, se puede crear la matriz adyacente del grafo M, poniendo

$$m_{ij} = \begin{cases} 1 & \text{si en la página } i \text{ hay un enlace de la página } j \\ 0 & \text{en caso contrario} \end{cases}$$

De esta manera, el PageRank implica

$$M^t \cdot x = \lambda \cdot x$$

donde λ es la constante de proporcionalidad y x es el vector que indica la importancia de la página que indica cada componente; es decir λ indica un valor propio

de M^t y x indica el vector propio de M^t asociado a λ.
Pero con esto, tenemos el problema que dicho λ no tiene porque ser real. Es por ello que dicha matriz M se perturba por $\widetilde{M} = c \cdot M + (1-c) \cdot U$ donde c es un parámetro entre 0 y 1 (Google usa $c = 0{,}85$ aproximadamente) y U es la matriz que tiene por elementos $\frac{1}{n}$, donde n indica la dimensión de M. De este modo, dicha matriz $\widetilde{M}$ si se puede asegurar que sus valores propios sean reales. Si dicha matriz tiene valor propio de módulo máximo, nos interesa que λ sea dicho valor, por lo cual caldria aplicar el método de la potencia a la matriz $\widetilde{M}$.

3.3. Iteración QR

Tal y como hemos visto en el 2.1.8 hay varios métodos para encontrar la descomposición QR de una matriz A, con R matriz triangular superior y Q ortogonal. Dicha factorización, nos servirá para calcular los valores propios.

Teorema 11.
Si A es una matriz $n \times n$ con valores propio λ_1, λ_2, λ_2, $\cdot$, λ_n entonces:

$$\prod_{i=1}^{n} \lambda_i = det(A)$$

Recordemos que en la descomposición QR, la matriz Q es ortogonal y la matriz R es triangular por lo que $det(Q) = 1$ y $det(R) = \prod_{i=1}^{n} r_{ii}$ respectivamente. Asi, haciendo la factorización QR a la matriz A y aplicando el teorema anterior obtenemos:

$$\prod_{i=1}^{n} \lambda_i = det(A) = det(QR) = det(Q) \cdot det(R) = 1 \cdot det(R) = det(R) = \prod_{i=1}^{n} r_{ii}$$

Con esto concluimos que dado un i $(1 \leq i \leq n)$, existirá un j $(1 \leq j \leq n)$ tal que $\lambda_i = r_{jj}$ con lo cual, los valores propio de A son los elementos de la diagonal principal de la matriz R

3.4. Descomposición en valores singulares (SVD)

Sea A una matriz $m \times n$ con $m \leq n$. Así $A^t \cdot A$ es una matriz $n \times n$

Definición 24.
Si $A^t \cdot A$ tiene valores propios λ_1, λ_2, λ_3, $\cdots$, λ_n entonces se llama valores singulares de A a $\sigma_1 = \sqrt{\lambda_1}$, $\sigma_2 = \sqrt{\lambda_2}$, $\sigma_3 = \sqrt{\lambda_3}$, $\cdots$, $\sigma_n = \sqrt{\lambda_n}$.

Una descomposición en valores singulares de A es una factorización del tipo $A = U \cdot \Sigma \cdot V^t$ con U matriz ortogonal $m \times m$, V matriz ortogonales $n \times n$ y Σ matriz diagonal $m \times n$.
La matriz Σ esta formada por los valores singulares de A en su diagonal principal

ordenador de mayor a menor. Es decir si $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n > 0$ son los valores propios de $A^t \cdot A$ y

$$\Sigma_n = \begin{pmatrix} \sqrt{\lambda_1} & 0 & 0 & \cdot & 0 \\ 0 & \sqrt{\lambda_2} & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{\lambda_3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{\lambda_n} \end{pmatrix}$$

entonces Σ se obtiene añadiendo columnas/filas de ceros a Σ_n hasta conseguir una matriz $m \times n$. La matriz V se obtiene poniendo en columnas los vectores propios de la matriz $A^t \cdot A$ ordenados igual que en Σ, es decir si v_1, v_2, $\cdots$, v_n son los vectores propios de valor propio λ_1, λ_2, λ_3, $\cdots$, λ_n (ordenados como antes) entonces:

$$V = \begin{pmatrix} v_1 & | & v_2 & | & v_3 & | & \cdots & | & v_n \end{pmatrix}$$

Finalmente, U se obtiene poniendo en columnas los vectores u_1, u_2, u_3, $\cdots$, u_m donde

$$u_i = \begin{cases} \frac{A \cdot v_i}{\sqrt{\lambda_1}} \text{ si } i = 1, \cdots, n \\ \\ 0 \text{ si } i = n+1, \cdots, m \end{cases}$$

3.5. Arnoldi

Este algoritmo obtiene una matriz H_n de dimensión $n \times n$ de la forma

$$H_n = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,n-1} & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,n-1} & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n,n-1} & h_{n,n} \end{pmatrix}$$

y una matriz Q_n de dimensión $n \times n$ de la forma

$$Q_n = \begin{pmatrix} q_1 & | & q_2 & | & q_3 & | & \cdots & | & q_n \end{pmatrix}$$

donde q_1, q_2, $\cdots$, q_n son vectores ortonormales llamados los vectores de Arnoldi tal que fijado k, los vectores q_1, q_2, $\cdots$, q_k genere el subespacio de Krylov $\mathcal{K}_k$ que satisface

$$H_n = Q_n^t \cdot A \cdot Q_n$$

o lo que es lo mismo

$$A \cdot Q_n = Q_{n+1} \cdot \widetilde{H}_n$$

con

$$\widetilde{H}_n = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \cdots & h_{1,n-1} & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \cdots & h_{2,n-1} & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \cdots & h_{3,n-1} & h_{3,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n,n-1} & h_{n,n} \\ 0 & 0 & 0 & \cdots & 0 & h_{n+1,n} \end{pmatrix}$$

Para hallar estas matrices se usa el algoritmo siguiente:
Dado un vector q arbitrario, definimos $q_1 = \frac{q}{||q||}$ y para $k = 2, 3, \cdots, n$

$$\begin{cases} q_k = A \cdot q_{k-1} \\ h_{j,k-1} = q_j^t \cdot q_k \ ; \ q_k = q_k - h_{j,k-1} \cdot q_j \text{ para } j = 1, \cdots, k \\ \\ h_{k,k-1} = ||q_k|| \\ q_k = \frac{q_k}{h_{k,k-1}} \end{cases}$$

3.6. GMRES (Generalized Minimum Residual)

Este algoritmo es una extensión del anterior pero para hallar la solución de un sistema $A \cdot x = b$. El algoritmo es el siguiente
Definimos $\beta = ||b||$, $q_1 = \frac{b}{\beta}$ y para $k = 2, 3, \cdots, n$

$$\begin{cases} q_k = A \cdot q_{k-1} \\ h_{j,k-1} = q_j^t \cdot q_k \ ; \ q_k = q_k - h_{j,k-1} \cdot q_j \text{ para } j = 1, \cdots, k \\ \\ h_{k,k-1} = ||q_k|| \\ \\ q_k = \frac{q_k}{h_{k,k-1}} \\ \\ \text{Encontramos } y_k \text{ tal que minimize la norma de } \widetilde{H}_k \cdot y_k - \beta \cdot e_1 \\ \\ x_k = Q_k \cdot y_k \end{cases}$$

donde $e_1 = (1, 0, \cdots, 0)$. Si la norma de $\widetilde{H}_k \cdot y_k - \beta \cdot e_1$ (con y_k el vector encontrado) es muy pequeño, el proceso finaliza. Para hallar este vector y_k lo que se hace es resolver el sistema $M \cdot y_k = c$ donde $M = \widetilde{H}_k^t \cdot \widetilde{H}_k$ y $c = \beta \cdot e_1$.

3.7. Lanczos

Este algoritmo transforma la matriz A del cual queremos hallar los valores propios en una matriz tridiagonal T de la forma

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \beta_2 & \alpha_2 & \beta_3 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \beta_3 & \alpha_3 & \beta_4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \alpha_{n-2} & \beta_{n-1} & 0 \\ 0 & 0 & 0 & 0 & \cdots & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & 0 & 0 & 0 & \cdots & 0 & \beta_n & \alpha_n \end{pmatrix} := \begin{pmatrix} \alpha_1 & 0 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \\ \vdots & \vdots \\ \alpha_{n-1} & \beta_{n-1} \\ \alpha_n & \beta_n \end{pmatrix}$$

del cual hallar los valores propios es más simple y son los mismos que A. Para hallar estos alfas y estos betas se usa el siguiente algoritmo:

Dado un vector b cualquiera, definimos $\beta_1 = 0$, $q_0 = 0$ y $q_1 = \frac{b}{||b||}$,

$$\begin{cases} v_i = A \cdot q_i - \beta_i \cdot q_{i-1} \\ \alpha_i = q_i^t \cdot v_i \\ v_i = v_i - \alpha_i \cdot q_i - \beta_i \cdot q_{i-1} \\ \beta_{i+1} = ||v_i|| \\ q_{i+1} = \frac{v_i}{\beta_{i+1}} \end{cases}$$

$\forall 1 \leq i \leq n-1$ y

$$\begin{cases} v_n = A \cdot q_n - \beta_n \cdot q_{n-1} \\ \alpha_n = q_n^t \cdot v_n \end{cases}$$

Una vez encontrada la matriz T se puede aplicar el teorema de Gerschgorin, que dice lo siguiente:

Teorema 12. *(Gerschgorin)*
Los valores propios de A están localizados, dentro del plano complejo, en la unión $\mathcal{F}$ de los discos

$$F_i = \left\{ z \in \mathbb{C},\ |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \right\}$$

para $1 \leq i \leq n$

En nuestro caso al ser la matriz T simetrica, los valores propios son todos reales. De forma que el teorema de Gerschgorin los discos se reducen a:

$$\begin{cases} F_1 = \{z \in \Re,\ |z - \alpha_1| \leq |\beta_2|\} \\ F_i = \{z \in \Re,\ |z - \alpha_i| \leq |\beta_i| + |\beta_{i+1}|\} \text{ para } 2 \leq i \leq n-1 \\ F_n = \{z \in \Re,\ |z - \alpha_n| \leq |\beta_n|\} \end{cases}$$

que esto es:

$$\begin{cases} F_1 = \{z \in \Re,\ \alpha_1 - |\beta_2| \leq z \leq \alpha_1 + |\beta_2|\} \\ F_i = \{z \in \Re,\ \alpha_i - |\beta_i| - |\beta_{i+1}| \leq z \leq \alpha_i + |\beta_i| + |\beta_{i+1}|\} \text{ para } 2 \leq i \leq n-1 \\ F_n = \{z \in \Re,\ \alpha_n - |\beta_n| \leq z \leq \alpha_n + |\beta_n|\} \end{cases}$$

Así los valores propios se encuentran en el intervalo (a, b) con $a = [x]-1$ y $b = [y]+1$ donde $[z]$ indica la parte entera del número z (es decir indica z sin la parte decimal; por ejemplo $[7{,}141516] = 7$),

$$x = \text{mín}\,\{\alpha_1 - |\beta_2|; \alpha_n - |\beta_n|; \alpha_i - |\beta_i| - |\beta_{i+1}|\}$$

$$y = \text{máx}\{\alpha_1 + |\beta_2|; \alpha_n + |\beta_n|; \alpha_i + |\beta_i| + |\beta_{i+1}|\}$$

Una vez encontrado el intervalo se puede encontrar fácilmente el polinomio caracteristico $p_n(x)$ de la matriz T mediante la recurrencia siguiente:

$$\begin{cases} p_0(x) = 1 \\ \\ p_1(x) = x - \alpha_1 \\ \\ p_i(x) = (x - \alpha_i) \cdot p_{i-1}(x) - \beta_i^2 \cdot p_{i-2}(x) \text{ para } 2 \leq i \leq n \end{cases}$$

Como $\{p_n(x), p_{n-1}(x), \cdots, p_1(x), p_0(x)\}$ forma una sucesión de Sturm, podemos usar el teorema de Sturm que dice lo siguiente:

Teorema 13. *(Sturm)*
Sea $\{f_0 = f, f_1, \cdots, f_n\}$ una sucesión de Sturm para f sobre $[c, d]$ con $f(c) \cdot f(d) \neq 0$; entonces el número de ceros reales de f en el intervalo (c, d) es igual a $V(c) - V(d)$, donde $V(t)$ es el número de cambios de signo que hay en total entre $f_{i-1}(x)$ y $f_i(x)$ para $1 \leq i \leq n$

Para ello dividimos el intervalo (a, b) en $(a, a+1)$, $(a+1, a+2)$, $\cdots$, $(b-1, b)$ y en cada uno de ellos aplicamos el teorema. Una vez localizados los intervalos podemos usar bisección en cada intervalo hasta encontrar todos los valores propios. Recordemos que el método de bisección sirve para hallar la solución de $f(x) = 0$. Se parte de un intervalo (a, b). Si $f(a)$ y $f(\frac{a+b}{2})$ son de signo contrario se repite con el intervalo $(a, \frac{a+b}{2})$, en caso contrario será $f(b)$ y $f(\frac{a+b}{2})$ quien tenga signo contrario y por tanto se repite con el intervalo $(\frac{a+b}{2}, b)$. A veces, puede ser que tanto $f(a)$ y $f(\frac{a+b}{2})$ como $f(b)$ y $f(\frac{a+b}{2})$ tengan diferente signo asi que se tendria que repetir el proceso con ambos intervalos del cual tendriamos una solución en $(a, \frac{a+b}{2})$ y otra en $\frac{a+b}{2}, b)$.

3.8. Aplicaciones

Tal y como dijimos en el capítulo anterior, un método iterativo $x^{(k+1)} = B \cdot x^{(k)} + c$ es convergente si $\rho(B) < 1$. En este apartado, usaremos los métodos vistos para hallar los valores propios o el radio espectral (según el método) de las matrices de Gauss-Seidel y Jacobi de los dos ejemplos del primer tema.

3.8.1. Potencia

Aplicando el método de la potencia a la matriz de Jacobi ($B_J = -D^{-1} \cdot (L + U)$) y Gauss-Seidel ($B_{GS} = -(D + L)^{-1} \cdot U$) de los dos ejemplos anteriores obtenemos que el radio espectral para ambas matrices son:

	$\rho(B_{GS})$		$\rho(B_J)$	
n	Ejemplo 1	Ejemplo 2	Ejemplo 1	Ejemplo 2
1	0,000000000000000	–	0,000000000000000	–
2	$7{,}499999999999999 \cdot 10^{-01}$	$2{,}500000000000001 \cdot 10^{-01}$	$-1{,}000000000000000$	$5{,}000000000000000 \cdot 10^{-01}$
3	$9{,}808589309964704 \cdot 10^{-01}$	$5{,}000000000002897 \cdot 10^{-01}$	$-1{,}722949669630174$	$7{,}058823529411763 \cdot 10^{-01}$
4	$9{,}990297924415606 \cdot 10^{-01}$	$6{,}545084971880498 \cdot 10^{-01}$	$-2{,}582091189278099$	$8{,}090169943748969 \cdot 10^{-01}$
5	$9{,}996483982620513 \cdot 10^{-01}$	$7{,}500000000014739 \cdot 10^{-01}$	$-3{,}444142191166121$	$8{,}659793814430697 \cdot 10^{-01}$
6	$9{,}999030990355475 \cdot 10^{-01}$	$8{,}117449009312795 \cdot 10^{-01}$	$-4{,}308531034793172$	$9{,}009688679019908 \cdot 10^{-01}$
7	$9{,}996734242041722 \cdot 10^{-01}$	$8{,}535533905964349 \cdot 10^{-01}$	$-5{,}174675493927592$	$9{,}238750041753874 \cdot 10^{-01}$
8	$9{,}994208557866564 \cdot 10^{-01}$	$8{,}830222215640230 \cdot 10^{-01}$	$-6{,}042134342076327$	$9{,}396926207847884 \cdot 10^{-01}$
9	$9{,}991556698075243 \cdot 10^{-01}$	$9{,}045084971933837 \cdot 10^{-01}$	$-6{,}910590171318841$	$9{,}510557630484175 \cdot 10^{-01}$
10	$9{,}990216154545027 \cdot 10^{-01}$	$9{,}206267664227856 \cdot 10^{-01}$	$-7{,}779815131930015$	$9{,}594929736125375 \cdot 10^{-01}$
20	$9{,}993161120312957 \cdot 10^{-01}$	$9{,}777864029211562 \cdot 10^{-01}$	$-1{,}649209898379260 \cdot 10$	$9{,}888308262147443 \cdot 10^{-01}$
30	$9{,}996631152707146 \cdot 10^{-01}$	$9{,}897649706903895 \cdot 10^{-01}$	$-2{,}521731007631978 \cdot 10$	$9{,}948693233685614 \cdot 10^{-01}$
40	$9{,}998625954734294 \cdot 10^{-01}$	$9{,}941402260819767 \cdot 10^{-01}$	$-3{,}394629803069707 \cdot 10$	$9{,}970658011421788 \cdot 10^{-01}$
50	$9{,}993709902981666 \cdot 10^{-01}$	$9{,}962103906844197 \cdot 10^{-01}$	$-4{,}267689509766453 \cdot 10$	$9{,}981033283250736 \cdot 10^{-01}$
60	$9{,}990744966599017 \cdot 10^{-01}$	$9{,}973503031511649 \cdot 10^{-01}$	$-5{,}140832562612850 \cdot 10$	$9{,}986740615000843 \cdot 10^{-01}$
70	$9{,}992334011741548 \cdot 10^{-01}$	$9{,}980439535113724 \cdot 10^{-01}$	$-6{,}014024340985873 \cdot 10$	$9{,}990208860126393 \cdot 10^{-01}$
80	$9{,}997046738074147 \cdot 10^{-01}$	$9{,}984970502105279 \cdot 10^{-01}$	$-6{,}887247065330224 \cdot 10$	$9{,}992463389567990 \cdot 10^{-01}$
90	$9{,}999763949271930 \cdot 10^{-01}$	$9{,}988090900548979 \cdot 10^{-01}$	$-7{,}760490667889259 \cdot 10$	$9{,}993996690594222 \cdot 10^{-01}$
100	$9{,}999668281835281 \cdot 10^{-01}$	$9{,}990328550165547 \cdot 10^{-01}$	$-8{,}633749020735333 \cdot 10$	$9{,}995073712786416 \cdot 10^{-01}$
1000	$9{,}994395427009232 \cdot 10^{-01}$	–	$-8{,}723195567151109 \cdot 10^{+02}$	–

Tabla 74: Radio espectral de las matrices de Gauss-Seidel y Jacobi de los dos ejemplos vistos

Así, por ejemplo para el primer ejemplo obtenemos el siguiente tiempo de ejecución:

Ejemplo 1		
n	Gauss-Seidel	Jacobi
1	1,888s	1,601s
2	0,355s	0,334s
3	0,310s	0,246s
4	0,279s	0,253s
5	0,255s	0,221s
6	0,254s	0,237s
7	0,255s	0,245s
8	0,263s	0,237s
9	0,263s	0,933s
10	1,082s	1,366s
20	1,130s	0,813s
30	0,811s	0,917s
40	0,790s	0,741s
50	0,830s	0,813s
60	0,786s	0,814s
70	1,194s	0,749s
80	0,950s	0,750s
90	0,962s	0,741s
100	1,105s	3,318s
1000	38,827s	1,496s

Tabla 75: Tiempo ejecución del método de la potencia para el primer ejemplo

3.8.2. GMRES

Usando GMRES al primer ejemplo anterior, resolviendo el sistema mediante el método del gradiente conjugado con 1000 iteraciones máximas y condición inicial x donde x es el vector que tiene por elementos $x_i = 0$, obtenemos lo siguiente:

En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1			
n	Error	n	Error
1	0,000000000000000	20	$2{,}368286034576172 \cdot 10^{-01}$
2	$6{,}647937228824600 \cdot 10^{-14}$	30	$2{,}717030513045356 \cdot 10^{-01}$
3	$2{,}642576155834430 \cdot 10^{-13}$	40	$1{,}578361297404817 \cdot 10^{-01}$
4	$3{,}953807124060238 \cdot 10^{-12}$	50	$2{,}704662454251113 \cdot 10^{-01}$
5	$2{,}264881154829348 \cdot 10^{-03}$	60	$2{,}255271563363609 \cdot 10^{-01}$
6	$4{,}832443968831893 \cdot 10^{-03}$	70	$1{,}997077068920817 \cdot 10^{-01}$
7	$1{,}588493238101228 \cdot 10^{-02}$	80	$5{,}096556971113729 \cdot 10^{-01}$
8	$9{,}484755894103452 \cdot 10^{-01}$	90	$1{,}696919756758962 \cdot 10^{-01}$
9	$8{,}932147904090950 \cdot 10^{-01}$	100	$1{,}266172378759705 \cdot 10^{-01}$
10	$3{,}926659846508307 \cdot 10^{-02}$	1000	$1{,}730095669031394 \cdot 10^{-01}$

Tabla 76: Error del primer ejemplo usando GMRES

3.8.3. Lanczos

Aplicando el método de Lanczos a la matriz A del primer ejemplo anterior obtenemos que el radio espectral para dicha matrices es:
Para dimensión 2 tenemos que en el intervalo $(0, 1)$ hay 2 valores propios.
Para dimensión 3 tenemos que en el intervalo $(0, 1)$ hay 3 valores propios.
Para dimensión 4 tenemos que en el intervalo $(0, 1)$ hay 4 valores propios.
Para dimensión 5 tenemos que en el intervalo $(0, 1)$ hay 2 valores propios y que en el intervalo $(1, 2)$ hay 3.
Para dimensión 6 tenemos que en el intervalo $(-1, 0)$ hay 1 valor propio, en el intervalo $(0, 1)$ hay 2 y en el intervalo $(1, 2)$ hay 3 valores propios.
Para dimensión 9 tenemos que el intervalo $(-5, -4)$ hay 1 valores propios, en el intervalo $(-1, 0)$ hay 1 valores propios, en el intervalo $(0, 1)$ hay 2 valores propios, en el intervalo $(1, 2)$ hay 3 valores propios, en el intervalo $(2, 3)$ hay 1 valores propios y en el intervalo $(6, 7)$ hay 1 valores propios.
Para dimensión 10 tenemos que en el intervalo $(-12, -11)$ hay 1 valores propios, en el intervalo $(-3, -2)$ hay 1 valores propios, en el intervalo $(-1, 0)$ hay 1 valores propios, en el intervalo $(0, 1)$ hay 2 valores propios, en el intervalo $(1, 2)$ hay 3 valores propios, en el intervalo $(3, 4)$ hay 1 valores propios y que en el intervalo $(12, 13)$ hay 1 valores propios.

3.9. Comentario

En el tema anterior vimos como algún método iterativo, perdía eficacia a partir de una cierta dimensión. A continuación, y a la vista de los resultados obtenido por los subapartados anteriores, vamos a explicar los motivos mirando los valores propios de las matrices correspondientes para ver si cumple o no la condición de convergencia de estos métodos iterativos.
Viendo la tabla de los valores propios de la matriz de Gauss-Seidel del primer ejemplo, observamos como cuanto mayor es la dimensión más cercano es a 1. Es por ello, que el error pierde precisión cuanto mayor es la dimensión. Por ser inferior a 1 el resultado sigue siendo aceptable pero por ser cada vez más cercano a 1, implica que el resultado es peor que con los métodos directos para dimensión pequeña pero algo mejor (por lo menos más aceptable) para dimensión grande. Mientras que en el segundo ejemplo, observamos como conforme aumenta la dimensión, el valor propio se acerca a 1, siendo a partir de $n = 40$ donde se acerca cada vez más, lo que empeora los resultados de dicho método.
En cambio, para la matriz de Jacobi del segundo ejemplo, observamos como para $n = 2$ obtenemos, en valor absoluto, un valor propio de 1, para $n = 3$ obtenemos, en valor absoluto, un valor propio de 1,859881393868167 y para n mayor que 3 el valor propio, en valor absoluto, va aumentando considerablemente alejándolo cada vez mas de 1 lo que, por la condición vista de la convergencia, hace que el método de Jacobi para dicho ejemplo no converja. Mientras que en el segundo ejemplo, observamos como conforme aumenta la dimensión, el valor propio se acerca a 1, siendo a partir de $n = 30$ donde se acerca cada vez más, lo que empeora los

resultados de dicho método.
Finalmente, para SOR, tenemos la convergencia del segundo ejemplo por ser A una matriz simétrica, definida positiva, real y tridiagonal a bloques, del cual podemos asegurar que el método converge cogiendo $w=\frac{2}{1+\sqrt{1-\rho(B_{GS})}}$, donde $\rho(B_{GS})=\rho(B_J)^2$ es el radio espectral de la matriz B del método de Gauss-Seidel. Para el primer ejemplo, podemos asegurar tambien la convergencia por ser A una matriz simétrica y definida positiva pero no disponemos de una fórmula que nos de este w óptimo. De este modo, gracias a como son las matrices de nuestros ejemplos no cal calcular los valores propios de la matriz de SOR para comprobar la convergencia.

4. Métodos numéricos de sistemas no lineales

Los siguientes métodos, especialmente, son útiles para sistemas no lineales pero disponen de una versión del cual permite usarlo para sistemas lineales. En dicho capítulo veremos dos métodos. Mientras que en el primer apartado veremos newton en diversas variables, en el segundo lo haremos con continuación.
Para los casos no lineales, tendremos un sistema F: $\Re^n \longrightarrow \Re^n$ del cual queremos encontrar $\overline{x} \in \Re^n$ tal que se cumpla $F(\overline{x})=\overrightarrow{0}$. Supongamos que $F \in \mathscr{C}^r$ con $r > 1$. En cambio, para los casos lineales, los aplicaremos para resolver los dos sistemas del primer capítulo.

4.1. Métodos de Newton en diversas variables

4.1.1. Caso no lineal

Dado un $\overrightarrow{x_0} \in \Re^n$ cercano a $\overline{x}$. Así pues existe un $\overrightarrow{h} \in \Re^n$ tal que $\overline{x} = \overrightarrow{x_0} + \overrightarrow{h}$ y $||h||$ sea pequeño. Haciendo Taylor a F obtenemos:

$$\overrightarrow{0} = F(\overline{x}) = F(\overrightarrow{x_0} + \overrightarrow{h}) = F(\overrightarrow{x_0}) + D_xF(\overrightarrow{x_0}) \cdot \overrightarrow{h} + o(||\overrightarrow{h}||^2).$$

Si $||\overrightarrow{h}||$ es pequeño, $||\overrightarrow{h}||^2$ aún es más pequeño. Así, se puede considerar $o(||\overrightarrow{h}||^2)$ como 0. De esta manera, la igualdad anterior queda:

$$\overrightarrow{0} = F(\overrightarrow{x_0}) + D_xF(\overrightarrow{x_0}) \cdot \overrightarrow{h} \Longrightarrow D_xF(\overrightarrow{x_0}) \cdot \overrightarrow{h} = -F(\overrightarrow{x_0}).$$

Sea $\overrightarrow{x_1}=\overrightarrow{x_0} - D_xF(\overrightarrow{x_0})^{-1} \cdot F(\overrightarrow{x_0})$. Si $\overrightarrow{x_0}$ es una aproximación de $\overline{x}$, $\overrightarrow{x_1}$ aún es una aproximación mejor. Repitiendo el proceso anterior obtenemos el siguiente método iterativo:

$$\boxed{\overrightarrow{x_{k+1}}=\overrightarrow{x_k} - D_xF(\overrightarrow{x_k})^{-1} \cdot F(\overrightarrow{x_k})}$$

Como calcular la inversa de una matriz puede ser muy costosa, el método es equivalente a:

$$\boxed{\begin{cases} D_xF(\overrightarrow{x_k}) \cdot \overrightarrow{h_k} = -F(\overrightarrow{x_k}) \Rightarrow \overrightarrow{h_k} = \cdots \text{ (resolviendo el sistema correspondiente)} \\ \overrightarrow{x_{k+1}} = \overrightarrow{x_k} + \overrightarrow{h_k} \end{cases}}$$

Supongamos que queremos encontrar los ceros de la función $f(x, y, z, w) = (x * x + 3 * y * y - z * z * z + w * w - 5, x * x * x - 2 * y * y - 10 * z + w, x * x + y * y * y + z * z - w + 20, x - y * y * y + z + w * w * w - 10)$. Aplicando Newton en diversas variables con condición inicial $(1, 1, 1)$ en 1000 iteraciones máximas, obtenemos como solución $(x, y, z, w) = (4{,}240052847770857, -3{,}901982902215025, 4{,}190960716777360, -3{,}867325957651389)$ en 21 iteraciones, del cual si lo sustituimos en nuestra f obtenemos $f(x, y, z, w) = (3{,}532504150305371 \cdot 10^{-14}, 4{,}798349217960407 \cdot 10^{-13}, 1{,}821459649775647 \cdot 10^{-14}, -1{,}476596622751458 \cdot 10^{-14})$, por lo cual concluimos que nos proporciona en dicho ejemplo un buen resultado.

4.1.2. Caso lineal

En tal caso, podemos definir F como $F(\overrightarrow{x}) = A \cdot \overrightarrow{x} - b$, $\overrightarrow{x} \in \Re^n$. De tal manera, $D_x F(\overrightarrow{x}) = A$. Así, el método de Newton en dicho caso sería:

$$\begin{cases} A \cdot \overrightarrow{h_k} = b - A \cdot \overrightarrow{x_k} \Rightarrow \overrightarrow{h_k} = \cdots \text{ (resolviendo el sistema correspondiente)} \\ \overrightarrow{x_{k+1}} = \overrightarrow{x_k} + \overrightarrow{h_k} \end{cases}$$

Aplicándolo a los dos ejemplos que teníamos en el primer tema (resolviendo el sistema según el método del gradiente conjugado con 1000 iteraciones máximas y condición inicial x, donde x es el vector que tiene por elementos $x_i = 0$ para ambos ejemplos) obtenemos los errores siguientes:
En el primer ejemplo obtenemos la tabla y gráfica siguiente:

Ejemplo 1					
n	Error	Iteraciones	n	Error	Iteraciones
1	0,000000000000000	1	20	$1{,}074408096138027 \cdot 10^{-04}$	1
2	$4{,}705040823708615 \cdot 10^{-15}$	1	30	$1{,}166242361102504 \cdot 10^{-04}$	1
3	$6{,}960574574405189 \cdot 10^{-13}$	1	40	$3{,}069107304188244 \cdot 10^{-04}$	1
4	$1{,}352021376164558 \cdot 10^{-12}$	1	50	$1{,}788047476559421 \cdot 10^{-04}$	1
5	$7{,}199076742080798 \cdot 10^{-12}$	1	60	$3{,}092498731559510 \cdot 10^{-04}$	1
6	$4{,}222677513233497 \cdot 10^{-10}$	1	70	$1{,}562333452586264 \cdot 10^{-04}$	1
7	$1{,}812082486813386 \cdot 10^{-04}$	1	80	$2{,}306302276351738 \cdot 10^{-04}$	1
8	$3{,}466431826041678 \cdot 10^{-05}$	1	90	$3{,}193729744393001 \cdot 10^{-04}$	1
9	$8{,}578988628576195 \cdot 10^{-05}$	1	100	$1{,}503886404778573 \cdot 10^{-04}$	1
10	$1{,}602553144942379 \cdot 10^{-04}$	1	1000	$5{,}105195490293124 \cdot 10^{-04}$	1

Tabla 77: Error del primer ejemplo usando Newton en diversas variables

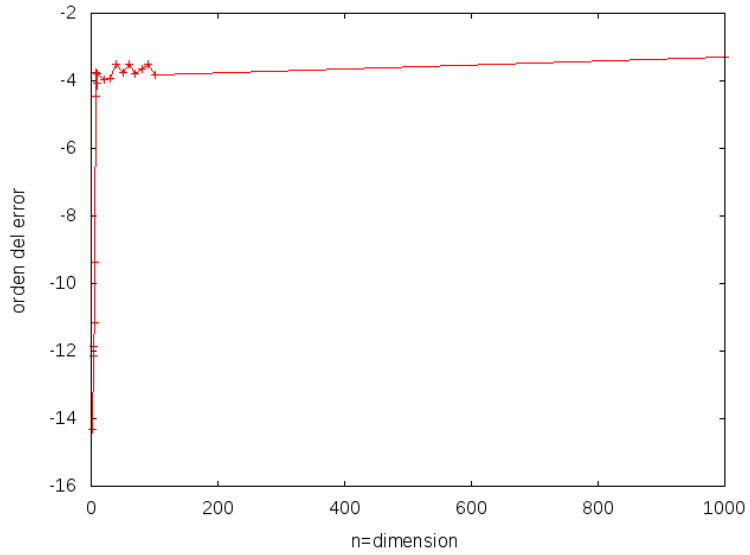


Figura 29: Error del primer ejemplo en escala logarítmica usando Newton en diversas variables

Mientras que en el segundo ejemplo lo que obtenemos es lo siguiente:

Ejemplo 2					
n^2	Error	Iteraciones	n^2	Error	Iteraciones
1	$3{,}568515041634690e-03$	1	20	$4{,}307202390722196e-04$	1
2	$2{,}755735094107417e-03$	1	30	$2{,}920293729433868e-04$	1
3	$2{,}158044791559726e-03$	1	40	$2{,}208714853580711e-04$	1
4	$1{,}758522566483584e-03$	1	50	$1{,}775894304169555e-04$	1
5	$1{,}479488029214253e-03$	1	60	$1{,}484883940223662e-04$	1
6	$1{,}275238805385523e-03$	1	70	$1{,}275807955015654e-04$	1
7	$1{,}119809922150506e-03$	1	80	$1{,}118336627727640e-04$	1
8	$9{,}977841775990082e-04$	1	90	$9{,}954644558721772e-05$	1
9	$8{,}995373727227325e-04$	1	100	$8{,}969178239823598e-05$	1
10	$8{,}187854975012863e-04$	1			

Tabla 78: Error del segundo ejemplo usando Newton en diversas variables

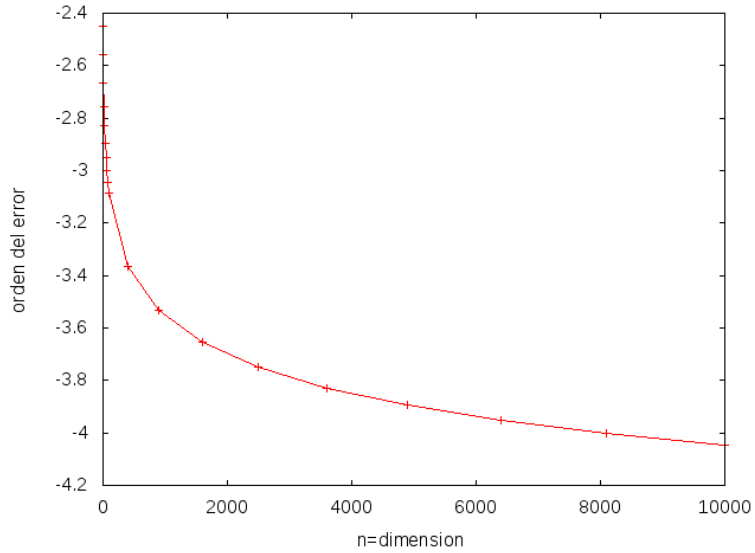


Figura 30: Error del segundo ejemplo en escala logarítmica usando Newton en diversas variables

Mirando las dos tablas podemos decir que con estos resultados, conforme aumenta la dimensión, la solución dada gana precisión. Según el contexto, el orden de 10^{-3} y 10^{-4} puede ser un error demasiado grande númericamente. Con esto podemos concluir que el método, para cualquier dimensión, es aceptable. Este método va siendo cada vez más aconsejable conforme aumenta la dimensión. Comparándolo con los métodos vistos, vemos que el método mejora a los anteriores en ambos ejemplos. Es por ello que podemos concluir que dicho método es el mejor de todos. El tiempo de ejecución para dicho método es:

	Newton en diversas variables	
n	Ejemplo 1	Ejemplo 2
1	1,514s	1,858s
2	0,285s	10,474s
3	0,246s	0,989s
4	0,253s	0,842s
5	0,333s	0,845s
6	0,293s	0,437s
7	0,310s	0,429s
8	0,309s	0,394s
9	0,285s	0,401s
10	1,221s	1,966s
20	0,853s	1,021s
30	0,765s	2,230s
40	0,901s	4,526s
50	0,822s	9,811s
60	0,869s	23,058s
70	0,858s	49,891s
80	0,866s	1m 35,030s
90	1,134s	2m 47,268s
100	1,420s	4m 47,456s
1000	2,208s	–

Tabla 79: Tiempo ejecución de Newton en diversas variables

Observación 10. *Se puede observar como Newton en diversas variables es el mejor método. Esto es lógico, ya que si se parte como condición inicial la solución de cualquiera de los métodos, dicho método la refina por lo cual no es de extrañar que mejore a cualquier método pero se parezca al del gradiente conjugado puesto que se parte de la condición inicial del gradiente conjugado, método más eficiente de los estudiado en el apartado 2.4*

4.2. Métodos de continuación

4.2.1. Caso no lineal

Supongamos que conocemos los ceros de una función G: $\Re^n \longrightarrow \Re^n$ cualquiera. En particular, dado un $x_0 \in \Re^n$ podemos definir $G(x) = F(x) - F(x_0)\ \forall x \in \Re^n$ donde obviamente x_0 es un cero de G.
Definimos la función H: $\Re^n \times [0,1] \longrightarrow \Re^n$ como

$$H(x, \lambda) = \lambda \cdot F(x) + (1 - \lambda) \cdot G(x)$$

(algunas veces el intervalo $[0,1]$ se extiende según la definición de H) de tal manera que conocemos un cero de $H(x,0)$ (según la función F a veces se puede definir H como la misma función F pero con el término independiente multiplicado por λ de tal modo que para $\lambda = 0$ queda una función sin término independiente del cual $\overrightarrow{0}$ es soluciónde $H(x,0)$) del cual, nuestro objetivo inicial es encontrar los ceros de $H(x,1)$. Sea N muy grande (por ejemplo $N = 500$). Defininimos $d = \frac{1}{N}$. El método de continuación es:

- Paso 1:
  Buscamos los ceros de $H(x, d)$ mediante el método de Newton en diversas variables con condición inicial el cero de G(x).

- Paso 2:
  Buscamos los ceros de $H(x, 2 \cdot d)$ mediante el método de Newton en diversas variables con condición inicial el cero encontrado en el paso 1.

- Paso 3:
  Buscamos los ceros de $H(x, 3 \cdot d)$ mediante el método de Newton en diversas variables con condición inicial el cero encontrado en el paso 2.

  $\vdots$

- Paso N:
  Buscamos los ceros de $H(x, N \cdot d)$=$H(x, 1)$ mediante el método de Newton en diversas variables con condición inicial el cero encontrado en el paso $N - 1$.

Por ejemplo, supongamos que queremos resolver $f(x) = x * x * x + x - 1$ o $g(x) = x * x + sin(x) + 1$. Definiendo $H(x, e) = x * x * x + x - e$ para el primer ejemplo y $H(x, e) = x * x + sin(x) + e$ para el segundo para $h = 0.001$ en 1000 iteraciones máximas obtenemos $x = 6{,}819104170798571 \cdot 10^{-01}$ para el primer ejemplo y $x = 5{,}811656390713902 \cdot 10^{-01}$ para el segundo. Sustituendo dicho valor en las funciones obtenemos $-9{,}999999999543449 \cdot 10^{-04}$ para el primero y 1,886752077850860 para el segundo. Así, se podría observar como el método es aceptable para el primer ejemplo pero empeora para el segundo.

4.2.2. Caso lineal

En tal caso, un ejemplo simple, sería definir H por:

$$H(x, \lambda) = (\lambda \cdot A + (1 - \lambda) \cdot Id) \cdot x - b$$

De este modo, para $\lambda = 0$ tenemos $x = b$ como solución y nuestro objetivo es encontrar los ceros de $H(x, 1)$. Para ello, podemos usar el método de continuación. Aplicándolo a los dos ejemplos del primer tema obtenemos:

Observación 11.
Dicho método lo que hace es ir refinando la solución. Es decir, se parte de una predicción que es la solución de $H(x, \lambda_{k-1})$ que se coge como condición inicial para refinar la solución de $H(x, \lambda_k)$ y de este modo corregirla.

5. Conclusiones

El objetivo de este trabajo era el estudio de los métodos de resolucińn de sistemas lineales y no lineales. Se han estudiado distintos métodos y aplicado a algunos ejemplos. Hemos podido observar que la idoneidad de un método u otro depende de distintos factores y del problema que se quiere resolver. En los ejemplos estudiados, se puede ver que el error es sensible a la dimensión del problema.

En algunos casos, como en el primer ejemplo, los métodos iterativos y de Krylov son mejores que los directos. En otros casos, como en el segundo ejemplo, es al revés, los métodos directos funcionan mejor que los iterativos y según el método que elijamos de Krylov también. Finalmente, tenemos un tercer y último caso, como sucede en los métodos QR, los cuales no son recomendables para dimensiones grandes por la cantidad de tiempo de ejecución que requieren y que en los casos estudiados no aportan buenos resultados.
Con los ejemplos estudiados, hemos observado que si hay que elegir un método eficaz para resolver estos sistemas ha de ser gradiente conjugado. Dicho método, tal y como hemos podido observar, no solo nos proporciona unos buenos resultados (puesto que el error es el más pequeño) sino que también lo hace en poco tiempo. Si queremos refinar la solución, podemos emplear Newton en diversas variables aunque la mejora que hemos obtenido es pequeña.
A continuación, vamos a comparar los errores cometidos y las gráficas por el gradiente conjugado y el siguiente método que nos proporciona buenos datos, que son SOR para el primero y Gauss (aunque también nos serviria Gauss con pivotaje, Gauss con pivotaje total, descomposición LU, descomposición LU con pivotaje o SOR).

Ejemplo 1				
	Gradiente conjugado		SOR	
n	Error	Iteraciones	Error	Iteraciones
1	0,000000000000000	1	0,000000000000000	1
2	$4{,}705040823708615 \cdot 10^{15}$	2	$9{,}171409881229778 \cdot 10^{-14}$	37
3	$6{,}960574574405189 \cdot 10^{13}$	3	$8{,}496887217546161 \cdot 10^{-13}$	195
4	$1{,}352021376164558 \cdot 10^{12}$	5	$1{,}674042999869109 \cdot 10^{-07}$	1000
5	$7{,}199076742080798 \cdot 10^{12}$	7	$1{,}834432197234958 \cdot 10^{-03}$	1000
6	$4{,}222677513233497 \cdot 10^{10}$	10	$1{,}478555850463555 \cdot 10^{-02}$	1000
7	$1{,}812082486813386 \cdot 10^{-04}$	9	$1{,}443187754363500 \cdot 10^{-02}$	1000
8	$3{,}466431826041678 \cdot 10^{-05}$	13	$1{,}863976010890975 \cdot 10^{-02}$	1000
9	$8{,}578988628576195 \cdot 10^{-05}$	13	$2{,}240387292929920 \cdot 10^{-02}$	1000
10	$1{,}602553144942379 \cdot 10^{-04}$	13	$2{,}070421120278589 \cdot 10^{-02}$	1000
20	$1{,}074408096138027 \cdot 10^{-04}$	19	$5{,}013493399254550 \cdot 10^{-02}$	1000
30	$1{,}166242361102504 \cdot 10^{-04}$	23	$7{,}256050170275473 \cdot 10^{-02}$	1000
40	$3{,}069107304188244 \cdot 10^{-04}$	22	$8{,}244046525687071 \cdot 10^{-02}$	1000
50	$1{,}788047476559421 \cdot 10^{-04}$	28	$8{,}720776149141425 \cdot 10^{-02}$	1000
60	$3{,}092498731559510 \cdot 10^{-04}$	29	$9{,}011087527975531 \cdot 10^{-02}$	1000
70	$1{,}562333452586264 \cdot 10^{-04}$	36	$9{,}223270840460189 \cdot 10^{-02}$	1000
80	$2{,}306302276351738 \cdot 10^{-04}$	34	$9{,}382380926114130 \cdot 10^{-02}$	1000
90	$3{,}193729744393001 \cdot 10^{-04}$	33	$9{,}520523667309923 \cdot 10^{-02}$	1000
100	$1{,}503886404778573 \cdot 10^{-04}$	42	$9{,}645057330109347 \cdot 10^{-02}$	1000
1000	$5{,}105195490293124 \cdot 10^{-04}$	84	$1{,}078437149461118 \cdot 10^{-01}$	1000

Tabla 80: Comparación del error del primer ejemplo entre Gradiente conjugado y SOR

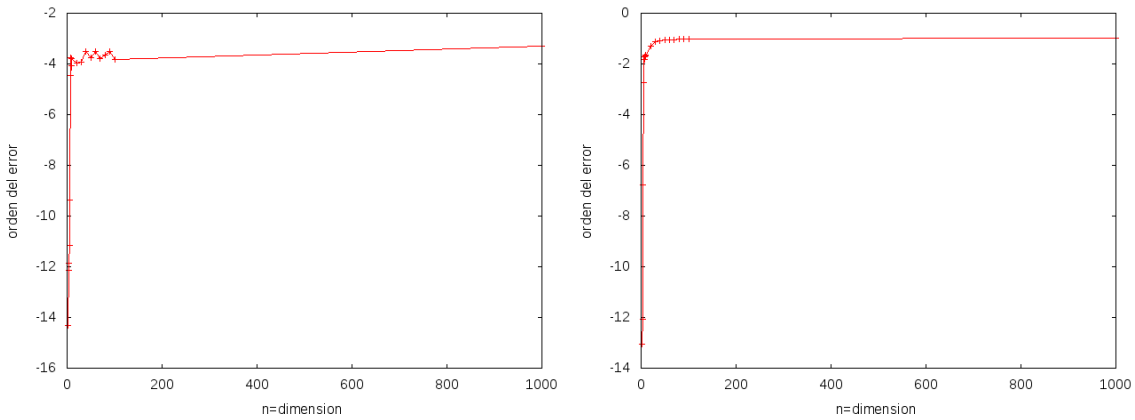


Figura 31: Comparación de las gráficas del primer ejemplo entre Gradiente conjugado y SOR

Tal y como se puede observar, en este primer ejemplo ambos métodos nos proporciona unos errores muy distantes, especialmente para dimensión grande. También, el número de iteraciones, dista bastante. Por lo cual, gradiente conjugado, en dicho ejemplo, mejora muchísimo.

Ejemplo 2			
	Gradiente conjugado		Gauss
n^2	Error	Iteraciones	Error
1	$3{,}568515041634690 \cdot 10^{-03}$	1	$3{,}568515041634690 \cdot 10^{-03}$
4	$2{,}755735094107417 \cdot 10^{-03}$	3	$2{,}755735094107197 \cdot 10^{-03}$
9	$2{,}158044791559726 \cdot 10^{-03}$	5	$2{,}158044791559693 \cdot 10^{-03}$
16	$1{,}758522566483584 \cdot 10^{-03}$	9	$1{,}758522566482695 \cdot 10^{-03}$
25	$1{,}479488029214253 \cdot 10^{-03}$	13	$1{,}479488029213543 \cdot 10^{-03}$
36	$1{,}275238805385523 \cdot 10^{-03}$	19	$1{,}275238805384803 \cdot 10^{-03}$
49	$1{,}119809922150506 \cdot 10^{-03}$	25	$1{,}119809922148691 \cdot 10^{-03}$
64	$9{,}977841775990082 \cdot 10^{-04}$	31	$9{,}977841776036746 \cdot 10^{-04}$
81	$8{,}995373727227325 \cdot 10^{-04}$	36	$8{,}995373727264779 \cdot 10^{-04}$
100	$8{,}187854975012863 \cdot 10^{-04}$	40	$8{,}187854975044966 \cdot 10^{-04}$
400	$4{,}307202390722196 \cdot 10^{-04}$	83	$4{,}307202391027821 \cdot 10^{-04}$
900	$2{,}920293729433868 \cdot 10^{-04}$	123	$2{,}920293663965129 \cdot 10^{-04}$
1600	$2{,}208714853580711 \cdot 10^{-04}$	163	$2{,}208714582493938 \cdot 10^{-04}$
2500	$1{,}775894304169555 \cdot 10^{-04}$	204	$1{,}775893530889277 \cdot 10^{-04}$
3600	$1{,}484883940223662 \cdot 10^{-04}$	243	$1{,}484882119725278 \cdot 10^{-04}$
4900	$1{,}275807955016151 \cdot 10^{-04}$	283	$1{,}275804263679398 \cdot 10^{-04}$
6400	$1{,}118336627728327 \cdot 10^{-04}$	323	$1{,}118330094669381 \cdot 10^{-04}$
8100	$9{,}954644558721323 \cdot 10^{-05}$	363	$9{,}954537040295337 \cdot 10^{-05}$
10000	$8{,}969178239819791 \cdot 10^{-05}$	403	$8{,}969010124756149 \cdot 10^{-05}$

Tabla 81: Comparación del error del segundo ejemplo entre Gradiente conjugado y Gauss

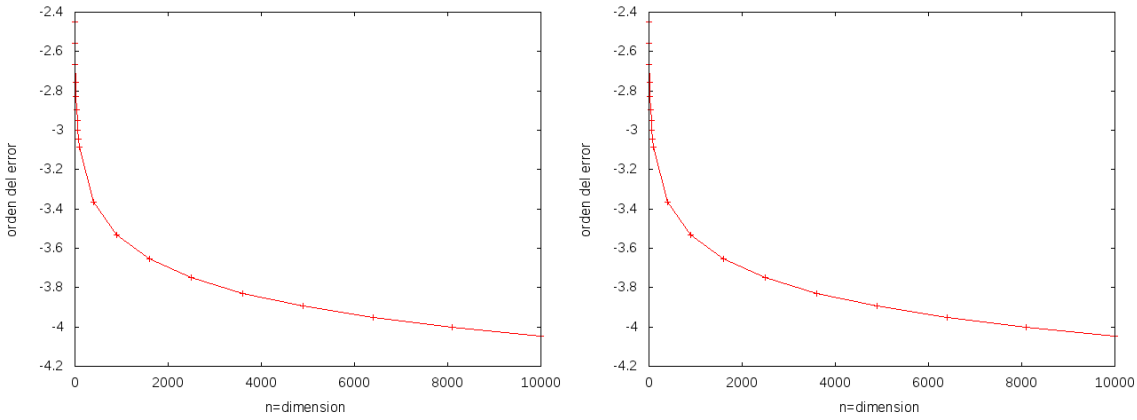


Figura 32: Comparación de las gráficas del segundo ejemplo entre Gradiente conjugado y Gauss

Tal y como se puede observar, en este segundo ejemplo ambos métodos nos proporcionan un error prácticamente idéntico. La decisión, por el tiempo de ejecución, favorecería a Gauss y cuanto al número de iterados, no es comparable pues Gauss al ser un método directo no tiene. Sin embargo, podríamos aumentar el número de iteraciones máximas de modo que Gradiente Conjugado mejorase los resultados y

el tiempo no afectase mucho (el tiempo depende básicamente del número de iteraciones máximas y de la dimensión y hay un teorema que dice que si un sistema $A \cdot x = b$ de dimensión $n \times n$ tal que A es simétrica y definida positiva entonces gradiente conjugado converge en un máximo de n iteraciones, por lo cual comparando las iteraciones con la dimensión, vemos que el tiempo no crecerá mucho), mientras que Gauss el error no tiene ninguna variable, por lo que siempre obtendríamos la misma tabla. Es por ello, que preferentemente se elije al Gradiente Conjugado.
Para los sistemas aleatorios tenemos que los resultados de cada método son prácticamente el mismo aunque alguno de ellos, como QR Householder, a pesar de darnos buenos resultados difieren un poco del resto. Es por ello que si los comparamos los resultados otenidos serán prácticamente el mismo para Crout, Doolittle, QR Givens, Gauss-Seidel, Jacobi, SOR y Gradiente conjugado.

Referencias

[1] Lloyd N. Trefethen; David Bau: *Numerical Linear Algebra*, Siam, 1997.

[2] James W. Demmel: *Applied Numerical Linear Algebra*, Siam, 1997.

[3] A. Aubanell; A. Benseny; A. Delshams: *Útiles Básicos de Cálculo Numérico*, Labor, 1993.

[4] John H. Mathews; Kurtis D. Fink: *Métodos Numérios con Matlab. Tercera edición*,Pearson Educación, 2000 (tema 4).

[5] Universidad de Salamanca [en linea] Última consulta: 16 de marzo del 2017. Disponible en
http : *//campus.usal.es/ mpg/PersonalesPersonalMAGL/Docencia/AlgebraTema4Teoria(*
11).*pdf*

[6] I Congreso Nacional de Estudiantes de Matemática [en linea] Última consulta: 7 de junio del 2017. Disponible en
https : *//atlas.mat.ub.edu/personals/dandrea/*2012_07_29_*google_corrientes.pdf*

Anexo I: Gauss

```
for(k=0; k<n-1; k++){
        if(fabs(a[k][k])<tol){
                        printf("Error: Estas dividiendo entre un
                        ↪  número muy pequeño \n");
        }
        for(i=k+1; i<n; i++){
                if(fabs(a[i][k])>tol){
                                m=a[i][k]/a[k][k];
                                for(j=k+1; j<n; j++){
                                        a[i][j]=a[i][j]-m*a[k][j];
                                }
                                b[i]=b[i]-m*b[k];
                }
        }
}
x[n-1]=b[n-1]/a[n-1][n-1];
for(i=n-2; i>=0; i--){
        m=0;
        for(j=i+1; j<n; j++){
                m=m+a[i][j]*x[j];
        }
        x[i]=(b[i]-m)/a[i][i];
}
```

Anexo II: Gauss con pivotaje

```
for(k=0; k<n-1; k++){
        r=k;
        for(i=k+1; i<n; i++){
                if(fabs(a[r][k])<=fabs(a[i][k])){
                        r=i;
                }
        }
        if(r!=k){
                for(i=0; i<n; i++){
                        m=a[k][i];
                        a[k][i]=a[r][i];
                        a[r][i]=m;
                }
                m=b[k];
                b[k]=b[r];
                b[r]=m;
        }
        if(fabs(a[k][k])<tol){
              printf("Error: Estas dividiendo entre un número muy
              ↪  pequeño\n");
```

```
          return;
        }
        for(i=k+1; i<n; i++){
                if(fabs(a[i][k])>tol){
                        m=a[i][k]/a[k][k];
                        for(j=k+1; j<n; j++){
                                a[i][j]=a[i][j]-m*a[k][j];
                        }
                        b[i]=b[i]-m*b[k];
                }
        }
}
for(i=n-1; i>=0; i--){
        m=0;
        for(j=i+1; j<n; j++){
                m=m+a[i][j]*x[j];
        }
        x[i]=(b[i]-m)/a[i][i];
}
```

Anexo III: Gauss con pivotaje total

```
for(i=0; i<n; i++){
        p[i][i]=1;
        for(j=0; j<i; j++){
                p[i][j]=0;
        }
        for(j=i+1; j<n; j++){
                p[i][j]=0;
        }
}
for(k=0; k<n-1; k++){
        r=k;
        s=k;
        for(i=k; i<n; i++){
                for(j=k; j<n; j++){
                        if(fabs(a[r][s])<fabs(a[i][j])){
                              r=i;
                              s=j;
                        }
                }
        }
        if(r!=k){
                for(i=k; i<n; i++){
                        m=a[k][i];
                        a[k][i]=a[r][i];
                        a[r][i]=m;
```

```
                }
                m=b[k];
                b[k]=b[r];
                b[r]=m;
        }
        if(s!=k){
                for(i=0; i<n; i++){
                        m=a[i][k];
                        a[i][k]=a[i][s];
                        a[i][s]=m;
                        m=p[i][k];
                        p[i][k]=p[i][s];
                        p[i][s]=m;
                }
        }
        if(fabs(a[k][k])<tol){
                printf("Error: Estas dividiendo entre un número muy
                 ↪  pequeño \n");
                return;
        }
        for(i=k+1; i<n; i++){
                if(fabs(a[i][k])>tol){
                        m=a[i][k]/a[k][k];
                        for(j=k+1; j<n; j++){
                                a[i][j]=a[i][j]-m*a[k][j];
                        }
                        b[i]=b[i]-m*b[k];
                }
        }
}
for(i=n-1; i>=0; i--){
        m=0;
        for(j=i+1; j<n; j++){
                m=m+a[i][j]*y[j];
        }
        y[i]=(b[i]-m)/a[i][i];
}
for(i=0; i<n; i++){
          for(j=0; j<n; j++){
                  if(p[i][j]==1){
                          x[i]=y[j];
                          j=n;
                  }
          }
}
```

Anexo IV: Descomposición LU

```
for(i=0; i<n; i++){
        l[i][i]=1;
        for(j=0; j<i; j++){
                l[i][j]=0;
        }
        for(j=i+1; j<n; j++){
                l[i][j]=0;
        }
}
for(k=0; k<n-1; k++){
        if(fabs(u[k][k])<tol){
                printf("Error: Estas dividiendo entre un número muy
                 ↪  pequeño\n");
                return;
        }
    for(i=k+1; i<n; i++){
            if(fabs(u[i][k])>tol){
                l[i][k]=u[i][k]/u[k][k];
            u[i][k]=0;
            for(j=k+1; j<n; j++){
                    u[i][j]=u[i][j]-l[i][k]*u[k][j];
            }
        }
        else{
                l[i][k]=0;
             u[i][k]=0;
        }
    }
}
y[0]=b[0];
for(k=1; k<n; k++){
        m=0;
        for(j=0; j<k; j++){
                m=m+l[k][j]*y[j];
        }
        y[k]=b[k]-m;
}
x[n-1]=y[n-1]/u[n-1][n-1];
for(i=n-2; i>=0; i--){
        m=0;
        for(j=i+1; j<n; j++){
                m=m+u[i][j]*x[j];
        }
        x[i]=(y[i]-m)/u[i][i];
}
```

Anexo V: Descomposición LU con pivotaje

```
for(i=0; i<n; i++){
        l[i][i]=1;
        for(j=0; j<i; j++){
                l[i][j]=0;
        }
        for(j=i+1; j<n; j++){
                l[i][j]=0;
        }
}
for(k=0; k<n-1; k++){
        r=k;
        for(i=k+1; i<n; i++){
                if(fabs(u[r][k])<=fabs(u[i][k])){
                        r=i;
                }
        }
        if(r!=k){
                for(i=0; i<n; i++){
              if(i<k){
                           m=l[k][i];
                           l[k][i]=l[r][i];
                           l[r][i]=m;
                   }
                   m=u[k][i];
              u[k][i]=u[r][i];
              u[r][i]=m;
             }
                m=b[k];
                b[k]=b[r];
                b[r]=m;
        }
        if(fabs(u[k][k])<tol){
                printf("Error: Estas dividiendo entre un número muy
                 ↪  pequeño \n");
                return;
        }
    for(i=k+1; i<n; i++){
           if(fabs(u[i][k])>tol){
                l[i][k]=u[i][k]/u[k][k];
                           u[i][k]=0;
                for(j=k+1; j<n; j++){
                           u[i][j]=u[i][j]-l[i][k]*u[k][j];
                }
        }
       else{
            l[i][k]=0;
```

```
                u[i][k]=0;
        }
    }
}
y[0]=b[0];
for(k=1; k<n; k++){
        m=0;
        for(j=0; j<k; j++){
                m=m+l[k][j]*y[j];
        }
        y[k]=b[k]-m;
}
x[n-1]=y[n-1]/u[n-1][n-1];
for(i=n-2; i>=0; i--){
        m=0;
        for(j=i+1; j<n; j++){
                m=m+u[i][j]*x[j];
        }
        x[i]=(y[i]-m)/u[i][i];
}
```

Anexo VI: Crout

```
for(k=0; k<n; k++){
        u[k][k]=1;
        for(i=k; i<n; i++){
              sum=0;
              for(j=0; j<k; j++){
                      sum=sum+l[i][j]*u[j][k];
              }
             l[i][k]=a[i][k]-sum;
        }
        if(fabs(l[k][k])<tol){
              printf("Error: Estas dividiendo entre un número muy
               ↪ pequeño \n");
              return;
        }
        for(i=k+1; i<n; i++){
              sum=0;
              for(j=0; j<k; j++){
                      sum=sum+l[k][j]*u[j][i];
              }
              u[k][i]=(a[k][i]-sum)/l[k][k];
        }
}
y[0]=b[0]/l[0][0];
for(k=1; k<n; k++){
```

```
        sum=0;
        for(j=0; j<k; j++){
                sum=sum+l[k][j]*y[j];
        }
        y[k]=(b[k]-sum)/l[k][k];
}
x[n-1]=y[n-1];
for(i=n-2; i>=0; i--){
        sum=0;
        for(j=i+1; j<n; j++){
                sum=sum+u[i][j]*x[j];
        }
        x[i]=y[i]-sum;
}
```

Anexo VII: Doolittle

```
for(k=0; k<n; k++){
        l[k][k]=1;
        for(i=k; i<n; i++){
              sum=0;
              for(j=0; j<k; j++){
                      sum=sum+l[k][j]*u[j][i];
              }
              u[k][i]=a[k][i]-sum;
        }
        if(fabs(u[k][k])<tol){
              printf("Error: Estas dividiendo entre un número muy
               ↪ pequeño \n");
              return;
        }
        for(i=k+1; i<n; i++){
              sum=0;
              for(j=0; j<k; j++){
                      sum=sum+l[i][j]*u[j][k];
              }
              l[i][k]=(a[i][k]-sum)/u[k][k];
        }
}
y[0]=b[0];
for(k=1; k<n; k++){
        sum=0;
        for(j=0; j<k; j++){
                sum=sum+l[k][j]*y[j];
        }
        y[k]=b[k]-sum;
}
```

```
x[n-1]=y[n-1]/u[n-1][n-1];
for(i=n-2; i>=0; i--){
        sum=0;
        for(j=i+1; j<n; j++){
                sum=sum+u[i][j]*x[j];
        }
        x[i]=(y[i]-sum)/u[i][i];
}
```

Anexo VIII: QR Householder

```
for(i=0; i<n; i++){
        q[i][i]=1;
        for(j=0; j<i; j++){
                q[i][j]=0;
        }
        for(j=i+1; j<n; j++){
                q[i][j]=0;
        }
}
for(k=0; k<n-1; k++){
        for(j=0; j<k; j++){
                u[j]=0;
        }
        for(j=k; j<n; j++){
                u[j]=r[j][k];
        }
        norm=0;
        for(j=0; j<n; j++){
                norm=norm+u[j]*u[j];
        }
        u[k]=u[k]-sqrt(norm);
        norm=0;
        for(j=0; j<n; j++){
                norm=norm+u[j]*u[j];
        }
        for(j=0; j<n; j++){
                u[j]=u[j]/sqrt(norm);
        }
        for(i=0; i<n; i++){
                p[i][i]=1-2*u[i]*u[i];
                for(j=0; j<i; j++){
                        p[i][j]=-2*u[i]*u[j];
                }
                for(j=i+1; j<n; j++){
                        p[i][j]=-2*u[i]*u[j];
                }
```

```
        }
        for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                        if(j==k && i>k){
                                a[i][j]=0;
                        }
                        else{
                                a[i][j]=0;
                                for(s=0; s<n; s++){
                                        a[i][j]=a[i][j]+p[i][s]*r[s][j];
                                }
                        }
                }
        }
        if(k==0){
                for(i=0; i<n; i++){
                        for(j=0; j<n; j++){
                                b[i][j]=p[i][j];
                        }
                }
        }
        else{
                for(i=0; i<n; i++){
                        for(j=0; j<n; j++){
                                b[i][j]=0;
                                for(s=0; s<n; s++){
                                        b[i][j]=b[i][j]+q[i][s]*p[s][j];
                                }
                        }
                }
        }
        for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                        r[i][j]=a[i][j];
                        q[i][j]=b[i][j];
                }
        }
}
for(i=0; i<n; i++){
        u[i]=0;
        for(j=0; j<n; j++){
                u[i]=u[i]+q[j][i]*v[j];
        }
}
x[n-1]=u[n-1]/r[n-1][n-1];
for(i=n-2; i>=0; i--){
```

```
        t=0;
        for(j=i+1; j<n; j++){
                t=t+r[i][j]*x[j];
        }
        x[i]=(u[i]-t)/r[i][i];
}
```

Anexo IX: QR Givens

```
for(i=0; i<n; i++){
        q[i][i]=1;
        a[i][i]=r[i][i];
        p[i][i]=q[i][i];
        for(j=0; j<i; j++){
                q[i][j]=0;
                a[i][j]=r[i][j];
                p[i][j]=q[i][j];
        }
        for(j=i+1; j<n; j++){
                q[i][j]=0;
                a[i][j]=r[i][j];
                p[i][j]=q[i][j];
        }
}
for(j=0; j<n-1; j++){
        for(i=n-1; i>j; i--){
                if(fabs(r[i][j])>tol){
                        c=r[i-1][j]/sqrt(r[i-1][j]*r[i-1][j]+r[i][j]*r[i][j]);
                        s=r[i][j]/sqrt(r[i-1][j]*r[i-1][j]+r[i][j]*r[i][j]);
                        for(k=0; k<n; k++){
                                a[i-1][k]=r[i-1][k]*c+r[i][k]*s;
                                a[i][k]=r[i-1][k]*s-r[i][k]*c;
                                p[i-1][k]=q[i-1][k]*c+q[i][k]*s;
                                p[i][k]=q[i-1][k]*s-q[i][k]*c;
                        }
                        a[i][j]=0;
                        for(k=0; k<n; k++){
                                for(l=0; l<n; l++){
                                        r[k][l]=a[k][l];
                                        q[k][l]=p[k][l];
                                }
                        }
                }
        }
}
for(i=0; i<n; i++){
        prod[i]=0;
```

```
        for(j=0; j<n; j++){
                prod[i]=prod[i]+q[i][j]*b[j];
        }
}
x[n-1]=prod[n-1]/r[n-1][n-1];
for(i=n-2; i>=0; i--){
        t=0;
        for(j=i+1; j<n; j++){
                t=t+r[i][j]*x[j];
        }
        x[i]=(prod[i]-t)/r[i][i];
}
```

Anexo X.1: Gauss-Seidel: Ejemplo 1

```
it=0;
do{
        for(i=0; i<n; i++){
                y[i]=x[i];
        }
        for(i=0; i<n; i++){
                sum=b[i];
                for(j=0; j<i; j++){
                        sum=sum-a[i][j]*x[j];
                }
        for(j=i+1; j<n; j++){
                        sum=sum-a[i][j]*x[j];
                }
                x[i]=sum/a[i][i];
        }
        norm=0;
        for(i=0; i<n; i++){
                norm=norm+(x[i]-y[i])*(x[i]-y[i]);
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo X.2: Gauss-Seidel: Ejemplo 2

```
it=0;
do{
        for(k=0; k<n*n; k++){
                y[k]=x[k];
        }
        k=0;
        x[k]=(x[k+1]+x[k+n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
```

```
                x[k]=(x[k+1]+x[k-1]+x[k+n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(x[k-1]+x[k+n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=(x[k+1]+x[k-n]+x[k+n]-b[k])/4.;
                k=k+1;
                for(j=2; j<n; j++){
                        x[k]=(x[k+1]+x[k-1]+x[k+n]+x[k-n]-b[k])/4.;
                        k=k+1;
                }
                x[k]=(x[k-1]+x[k-n]+x[k+n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(x[k+1]+x[k-n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=(x[k+1]+x[k-1]+x[k-n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(x[k-1]+x[k-n]-b[k])/4.;
        norm=0;
        for(k=0; k<n*n; k++){
                norm=norm+(x[k]-y[k])*(x[k]-y[k]);
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo XI.1: Jacobi: Ejemplo 1

```
it=0;
do{
        for(i=0; i<n; i++){
                y[i]=x[i];
        }
        for(i=0; i<n; i++){
                sum=b[i];
                for(j=0; j<i; j++){
                        sum=sum-a[i][j]*y[j];
                }
        for(j=i+1; j<n; j++){
                        sum=sum-a[i][j]*y[j];
                }
                x[i]=sum/a[i][i];
        }
        norm=0;
```

```
        for(i=0; i<n; i++){
                norm=norm+(x[i]-y[i])*(x[i]-y[i]);
        }
        it=it+1;
} while(sqrt(norm)>=tol && it<itmax);
```

Anexo XI.2: Jacobi: Ejemplo 2

```
it=0;
do{
        for(k=0; k<n*n; k++){
                y[k]=x[k];
        }
        k=0;
        x[k]=(y[k+1]+y[k+n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=(y[k+1]+y[k-1]+y[k+n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(y[k-1]+y[k+n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=(y[k+1]+y[k-n]+y[k+n]-b[k])/4.;
                k=k+1;
                for(j=2; j<n; j++){
                        x[k]=(y[k+1]+y[k-1]+y[k+n]+y[k-n]-b[k])/4.;
                        k=k+1;
                }
                x[k]=(y[k-1]+y[k-n]+y[k+n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(y[k+1]+y[k-n]-b[k])/4.;
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=(y[k+1]+y[k-1]+y[k-n]-b[k])/4.;
                k=k+1;
        }
        x[k]=(y[k-1]+y[k-n]-b[k])/4.;
        norm=0;
        for(k=0; k<n*n; k++){
                norm=norm+(x[k]-y[k])*(x[k]-y[k]);
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo XII.1: SOR: Ejemplo 1

```
it=0;
do{
        for(i=0; i<n; i++){
                y[i]=x[i];
        }
        for(i=0; i<n; i++){
                sum=b[i];
                for(j=0; j<n; j++){
                        sum=sum-a[i][j]*x[j];
                }
                x[i]=x[i]+w*sum*1./a[i][i];
        }
        norm=0;
        for(i=0; i<n; i++){
                norm=norm+(x[i]-y[i])*(x[i]-y[i]);
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo XII.2: SOR: Ejemplo 2

```
it=0;
do{
        for(k=0; k<n*n; k++){
                y[k]=x[k];
        }
        k=0;
        x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k+n]-b[k]);
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k-1]+x[k+n]-b[k]);
                k=k+1;
        }
        x[k]=x[k]+w/4*(-4*x[k]+x[k-1]+x[k+n]-b[k]);
        k=k+1;
        for(i=2; i<n; i++){
                x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k-n]+x[k+n]-b[k]);
                k=k+1;
                for(j=2; j<n; j++){
                        x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k-1]+x[k+n]+x[k-n]-b[k]);
                        k=k+1;
                }
                x[k]=x[k]+w/4*(-4*x[k]+x[k-1]+x[k-n]+x[k+n]-b[k]);
                k=k+1;
        }
        x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k-n]-b[k]);
        k=k+1;
```

```
        for(i=2; i<n; i++){
                x[k]=x[k]+w/4*(-4*x[k]+x[k+1]+x[k-1]+x[k-n]-b[k]);
                k=k+1;
                }
                x[k]=x[k]+w/4*(-4*x[k]+x[k-1]+x[k-n]-b[k]);
        }
        norm=0;
        for(k=0; k<n*n; k++){
                norm=norm+(x[k]-y[k])*(x[k]-y[k]);
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo XIII: Minimización

```
for(i=0; i<n; i++){
        p[i]=-b[i];
        for(j=0; j<n; j++){
                p[i]=p[i]+a[i][j]*x[j];
        }
}
it=0;
do{
        for(i=0; i<n; i++){
                y[i]=x[i];
        }
        m=0;
        for(i=0; i<n; i++){
                m=m+p[i]*p[i];
        }
        for(i=0; i<n; i++){
                h[i]=0;
                for(j=0; j<n; j++){
                        h[i]=h[i]+a[j][i]*p[j];
                }
        }
        t=0;
        for(i=0; i<n; i++){
                t=t+h[i]*p[i];
        }
        if(t==0){
                return(it);
        }
        al=m/t;
        for(i=0; i<n; i++){
                x[i]=x[i]-al*p[i];
        }
```

```
        for(i=0; i<n; i++){
                p[i]=-b[i];
                for(j=0; j<n; j++){
                        p[i]=p[i]+a[i][j]*x[j];
                }
        }
        normx=0;
        for(i=0; i<n; i++){
                normx=normx+(x[i]-y[i])*(x[i]-y[i]);
        }
        normf=0;
        for(i=0; i<n; i++){
                normf=normf+p[i]*p[i];
        }
        it=it+1;
}while(sqrt(normx)>tol && fabs(normf)>tol && it<itmax);
```

Anexo XIV: Gradiente conjugado

```
it=0;
for(i=0; i<n; i++){
        r[i]=b[i];
        for(j=0; j<n; j++){
                r[i]=r[i]-a[i][j]*x[j];
        }
        d[i]=r[i];
}
norm=0;
for(i=0; i<n; i++){
        norm=norm+r[i]*r[i];
}
do{
        y=0;
        for(i=0; i<n; i++){
                z[i]=0;
                for(j=0; j<n; j++){
                        z[i]=z[i]+a[i][j]*d[j];
                }
                y=y+d[i]*z[i];
        }
        if(y==0){
                return(it);
        }
        p=norm/y;
        for(i=0; i<n; i++){
                x[i]=x[i]+p*d[i];
                r[i]=r[i]-p*z[i];
```

```
        }
        c=norm;
        if(c==0){
                return(it);
        }
        norm=0;
        for(i=0; i<n; i++){
                norm=norm+r[i]*r[i];
        }
        al=norm/c;
        for(i=0; i<n; i++){
                d[i]=r[i]+al*d[i];
        }
        it=it+1;
} while(sqrt(norm)>=tol  && it<itmax);
```

Anexo XV.1: Matriz y vector: Ejemplo 1

```
for(i=0; i<n; i++){
        b[i]=0;
        for(j=0; j<n; j++){
                a[i][j]=1./(i+j+1);
        b[i]=b[i]+a[i][j];
        }
}
```

Anexo XV.2: Matriz y vector: Ejemplo 2

```
b[k]=h*f(1./(n+1),1./(n+1))-u(0,1./(n+1))-u(1./(n+1),0);
a[k][k]=-4;
a[k][k+1]=1.;
a[k][k+n]=1;
k=k+1;
for(j=2;j<n; j++){
        b[k]=h*f(1./(n+1),j/(n+1.))-u(0,j/(n+1.));
        a[k][k]=-4;
        a[k][k+1]=1.;
        a[k][k-1]=1.;
        a[k][k+n]=1;
        k=k+1;
}
b[k]=h*f(1./(n+1),n/(n+1.))-u(0,n/(n+1.))-u(1./(n+1),1);
a[k][k]=-4;
a[k][k-1]=1.;
a[k][k+n]=1;
k=k+1;
for(i=2; i<n; i++){
        b[k]=h*f(i/(n+1.),1./(n+1))-u(i/(n+1.),0);
```

```
        a[k][k]=-4;
        a[k][k+1]=1.;
        a[k][k-n]=1.;
        a[k][k+n]=1;
        k=k+1;
        for(j=2; j<n; j++){
                b[k]=h*f(i/(n+1.),j/(n+1.));
                a[k][k+1]=1.;
                a[k][k+n]=1.;
                a[k][k]=-4.;
                a[k][k-1]=1.;
                a[k][k-n]=1.;
                k=k+1;
        }
        b[k]=h*f(i/(n+1.),n/(n+1.))-u(i/(n+1.),1);
        a[k][k]=-4;
        a[k][k-1]=1.;
        a[k][k+n]=1;
        a[k][k-n]=1;
        k=k+1;
}
b[k]=h*f(n/(n+1.),1./(n+1))-u(1,1./(n+1))-u(n/(n+1.),0);  // Aqui
 ↪  habia un error
a[k][k]=-4;
a[k][k+1]=1.;
a[k][k-n]=1;
k=k+1;
for(j=2;j<n; j++){
        b[k]=h*f(n/(n+1.),j/(n+1.))-u(1,j/(n+1.));
        a[k][k]=-4;
        a[k][k+1]=1.;
        a[k][k-1]=1.;
        a[k][k-n]=1;
        k=k+1;
}
b[k]=h*f(n/(n+1.),n/(n+1.))-u(1,n/(n+1.))-u(n/(n+1.),1);
a[k][k]=-4;
a[k][k-1]=1.;
a[k][k-n]=1;
```

Anexo XVI: Sistemas aleatorios

```
conmat=drand48()*15+1; // generar el número de condición de la
 ↪  matriz A aletoriamente (random)
do{
        d[0]=drand48(); // generar el valor propio más pequeño de A
         ↪  aleatoriamente (random)
```

```
}while(d[0]<tol);
d[n-1]=d[0]/conmat;
for(i=1;i<n-1; i++){
        num=(d[n-i]+d[0])/2.;
        do{
                d[n-i-1]=drand48()*(num-d[n-i])+d[n-i]; // generar
                 ↪  el valor propio i-ésimo más pequeño de A
                 ↪  aleatoriamente entre d[i-1] y d[n-1] (o entre
                 ↪  d[i-1] y (d[i-1]+d[n-1])/2) (random)
        }while(d[n-i-1]<tol);
}
for(i=0; i<n; i++){
        for(j=0; j<n; j++){
                m[i][j]=rand()\%100; // generar la matriz M
                 ↪  aleatoriamente (random) con elementos de 0 a 99
        }
        b[i]=rand()\%100; // generar el vector de terminos
         ↪  independite b aleatoriamente (random) con elementos de 0
         ↪  a 99
}
descqr(m, q, n);
for(i=0; i<n; i++){
        for(j=0; j<n; j++){
                a[i][j]=0;
                for(k=0; k<n; k++){
                        a[i][j]=a[i][j]+d[k]*q[i][k]*q[j][k];
                }
        }
}
```